NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MATLAB GUI FOR A LEGENDRE PSEUDOSPECTRAL ALGORITHM FOR OPTIMAL CONTROL PROBLEMS

by

Andrew O. Hall

June 1999

Advisor: Fariba Fahroo
Second Reader: I. Michael Ross

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)

2. REPORT DATE
June, 1999

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE A MATLAB GUI FOR A LEGENDRE PSEUDO-SPECTRAL ALGORITHM FOR OPTIMAL CONTROL PROBLEMS

5. FUNDING NUMBERS

6. AUTHORS Hall, Andrew O.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT(*maximum 200 words*)

This implementation of a Legendre-Gauss-Lobatto Pseudospectral (LGLP) algorithm takes advantage of the MATLAB Graphical User Interface (GUI) and the Optimization Toolbox to allow an efficient implementation of a direct solution technique. Direct solutions techniques solve optimal control problems without solving for the optimality conditions. Using the LGLP method, an optimal control problem is discretized into a Nonlinear Program (NLP) and solved using an NLP solver, avoiding the problems of deriving the conditions of optimality and solving the resulting boundary value problem. The MATLAB GUI implementation solves optimal control problems without requiring knowledge of the specific implementation of the LGLP method. The GUI completes the discretization of the problem and solves the resulting NLP using a Sequential Quadratic Programming Algorithm. The GUI will convert any optimal control problem with fixed, free or optimal final time, a Mayer, Lagrange or Bolza cost function, constrained or unconstrained controls, with or without state inequalities, and point inequalities into a parameter optimization problem and returns a solution. The GUI creates a function file, output file, binary save file, and optimization script to allow full access to the strength of the LGLP method from the GUI or the command line. No prior knowledge of the LGLP algorithm is assumed or necessary.

14. SUBJECT TERMS Direct Methods, Optimal Control Theory, Calculus of Variations MATLAB, Nonlinear Programming, Optimization

15. NUMBER OF PAGES 100

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited

A MATLAB GUI FOR A LEGENDRE PSEUDOSPECTRAL ALGORITHM FOR OPTIMAL CONTROL PROBLEMS

Andrew O. Hall
Captain, United States Army
B.S., United States Military Academy, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 1999

Author: Andrew O. Hall

Andrew O. Hall

Approved by: Fariba Fahroo

Fariba Fahroo, Advisor

I. Michael Ross, Second Reader

M. A. Morgan

Michael Morgan, Chairman
Department of Mathematics

ABSTRACT

This implementation of a Legendre-Gauss-Lobatto Pseudospectral (LGLP) algorithm takes advantage of the MATLAB Graphical User Interface (GUI) and the Optimization Toolbox to allow an efficient implementation of a direct solution technique. Direct solutions techniques solve optimal control problems without solving for the optimality conditions. Using the LGLP method, an optimal control problem is discretized into a Nonlinear Program (NLP) and solved using an NLP solver, avoiding the problems of deriving the conditions of optimality and solving the resulting boundary value problem. The MATLAB GUI implementation solves optimal control problems without requiring knowledge of the specific implementation of the LGLP method. The GUI completes the discretization of the problem and solves the resulting NLP using a Sequential Quadratic Programming Algorithm. The GUI will convert any optimal control problem with fixed, free or optimal final time, a Mayer, Lagrange or Bolza cost function, constrained or unconstrained controls, with or without state inequalities, and point inequalities into a parameter optimization problem and returns a solution. The GUI creates a function file, output file, binary save file, and optimization script to allow full access to the strength of the LGLP method from the GUI or the command line. No prior knowledge of the LGLP algorithm is assumed or necessary.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Optimal Control Theory is one of the modern applications of the Calculus of Variations. Motivated by the development of modern computers and the space program during the 1960s, the theory of the calculus of variations has been used to create methods for designing modern systems. In 1962 Pontryagin's minimum principle gave the theoretical basis for solving optimal control problems. [Ref. 2]

The calculus of variations was developed to describe situations that nature had already optimized. The theory seeks to minimize or maximize the value of integrals. A classic example, known as the brachistochrone problem [Ref. 1], is to minimize the time a bead takes to slide down a wire between two points. In this problem, the shape of the wire can be directly changed to decrease the time the bead requires to move along the wire. In this type of problem, the physical states can be directly altered. The system can be represented as finding y, the shape of the wire, so that the time, expressed as an integral of the physical states, can be minimized. The following expression

$$\mathcal{J}(\mathbf{x},\mathbf{y}) = \int_a^b \mathbf{f}(\mathbf{x},\mathbf{y},\dot{\mathbf{y}})d\mathbf{x} \tag{1.1}$$

represents the cost function to be minimized.[1]

Optimal Control Theory is a generalization of the calculus of variations, but the control problems have unique characteristics that identify this class of problem. Optimal Control problems select from available controls to optimize the performance of a dynamic system. These problems focus on altering forces at our disposal to control a process and optimize some performance measure of the system [Ref. 1]. The controls can be altered, but the state of the system may not be directly altered. If $\dot{\mathbf{y}} = \mathbf{u}$ is used as a state equation and substituted into equation (1.1), the calculus

[1]Throughout this thesis boldface notation will be used to distinguish $n \times 1$ vectors or functions of $n \times 1$ vectors.

of variations problem can be converted into the following optimal control problem,

$$\mathcal{J}(\mathbf{x}, \mathbf{y}, \mathbf{u}) = \int_a^b \mathbf{f}(\mathbf{x}, \mathbf{y}, \mathbf{u}) d\mathbf{x}. \qquad \text{[Ref. 1]} \tag{1.2}$$

The emphasis in optimal control problems is on the controlling process. The adjustable variables are the control variables, not the state variables.

Optimal control problems also add constraints to the calculus of variations. Controls might have physical limitations, such as the force from an aircraft engine or the length of a robotic arm, and these constraints are modelled into the dynamic system that must be optimized. These types of problems in applied mathematics lead to engineering applications where controlling devices can be used to guide the state equations in an optimal way. The historical motivations for the field were the space and rocket programs and problems, such as the lunar landings. As the field developed, it has been applied to financial, robotic, artificial intelligence and other engineering applications.

Optimal Control Theory seeks to control a system while minimizing or maximizing a cost function. The cost function will reflect the change in some state of the system, some measurable amount of time, or the amount of control effort required for the system to reach a desired end state. Finding optimal functions is the goal of optimal control theory. This differs from standard parameter optimization because with optimal control theory, optimization is performed over a continuous function space.

In optimal control problems, dynamic constraints are added to the cost function to be optimized. The state equations and inequality constraints on the controls and states are adjoined to the cost function through the use of Lagrange multipliers. The Hamiltonian is used to simplify the equations used in this minimization. The Hamiltonian represents the sum of potential and kinetic energies used in a dynamic system. The first variations of the cost function components are set to zero to ensure a local minimization of the cost function. The equations that result are the necessary conditions for optimality and are referred to as the Euler-Lagrange equations. An example of an augmented cost function, $\mathcal{J}_A$, is

$$\mathcal{J}_A = \phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} \left\{ \mathcal{L}(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^T [\mathbf{f}(\mathbf{x}, \mathbf{u}, t) - \dot{\mathbf{x}}] + \boldsymbol{\mu}^T \mathbf{g}(\mathbf{u}, t) \right\} dt, \tag{1.3}$$

with a dynamic constraint of $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$, initial condition of $\mathbf{x}(t_0) = \mathbf{x}_0$, and equality constraints of $\mathbf{g}(\mathbf{u}, t) = 0$. This augmented Bolza cost function has a final time cost, $\phi(\mathbf{x}(t_f), t_f)$, and an integral cost, $\int_{t_0}^{t_f} \mathcal{L}(\mathbf{x}, \mathbf{u}, t)$, which are adjoined by the state constraints. The variables $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are Lagrange multipliers. The augmented Hamiltonian is defined as

$$\mathcal{H} = \mathcal{L} + \boldsymbol{\lambda}^T \mathbf{f} + \boldsymbol{\mu}^T \mathbf{g}. \tag{1.4}$$

The optimality conditions are[2]

$$\boldsymbol{\lambda}(t_f) = \left[\frac{\partial \phi}{\partial \mathbf{x}}\right]\Big|_{t=t_f} \tag{1.5}$$

$$\begin{aligned} \dot{\boldsymbol{\lambda}} &= -\left(\frac{\partial \mathcal{L}}{\partial \mathbf{x}}\right) - \left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)^T \boldsymbol{\lambda} \\ &= -\left(\frac{\partial \mathcal{H}}{\partial \mathbf{x}}\right) \end{aligned} \tag{1.6}$$

$$\begin{aligned} \mathbf{0} &= -\left(\frac{\partial \mathcal{L}}{\partial \mathbf{u}}\right) - \left(\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right)^T \boldsymbol{\lambda} - \left(\frac{\partial \mathbf{g}}{\partial \mathbf{u}}\right)^T \boldsymbol{\mu} \\ &= -\left(\frac{\partial \mathcal{H}}{\partial \mathbf{u}}\right). \end{aligned} \tag{1.7}$$

The solutions to the derived optimality conditions often rely on numerical methods for solving two point boundary value problems.

Optimal control algorithms can be divided into two types, direct and indirect methods. Most texts on optimal control theory, [Ref. 1], [Ref. 2], [Ref. 6], [Ref. 14], focus on indirect numerical techniques. Indirect methods rely on the necessary optimality conditions derived from the minimum principle. The necessary conditions are developed and then solved to find an optimal trajectory. The solution normally

[2] The gradient and Jacobian used to develop the optimal conditions will be defined in the next section.

requires solving a nonlinear two point boundary value problem (BVP), and solving for the Lagrange multipliers and costates. Nonlinear BVPs normally do not have a closed form solution and theorems do not exist that guarantee existence and uniqueness for all BVPs. This lack of an analytic solution requires the use of numerical methods to solve the BVP. The most common indirect method in use today is an indirect shooting algorithm [Ref. 3]. Direct methods discretize the continuous problem into a parameter optimization problem and then solve the resulting nonlinear optimization problem. Direct methods do not explicitly employ the necessary conditions for optimality, but the results can be checked using the optimality conditions from the calculus of variations.

An advantage of indirect methods is that they directly solve the adjoint differential equations for the conditions derived from the minimum principle and the transversality conditions to ensure optimality. The first derivative of the Hamiltonian is set to zero and the resulting system of ordinary differential equations is solved. This advantage leads to several nontrivial problems:

1. Necessary conditions must be derived analytically.
2. Region of convergence for root finding algorithms required to solve the BVP may be very small requiring a very "good" initial guess.
3. Path inequalities may require solving for constrained and unconstrained sub arcs before each iteration begins.
4. When setting the gradient to zero, analytic expressions for the gradient must be calculated.

There are several indirect numeric methods for optimal control problems. One such method is the *neighboring extremal* method or *shooting* method [Ref. 6]. Starting values for the state vector, $x_{(t_0)}$, and adjoint vector $\boldsymbol{\lambda}(t_0)$, are guessed and then the state equations

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{1.8}$$

and costate equations

$$\dot{\boldsymbol{\lambda}}(t) = -\left[\frac{\partial \mathcal{H}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, t)}{\partial \mathbf{x}}\right] \tag{1.9}$$

are integrated from t_0 to t_f with the control history

$$-\left[\frac{\partial \mathcal{H}(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, t)}{\partial \mathbf{u}}\right] = \mathbf{0}. \tag{1.10}$$

The optimal trajectory that results from the solution of the differential algebraic equation does not necessarily satisfy the end conditions. The starting values of the adjoint variable, $\boldsymbol{\lambda}$, can be perturbed in various ways to meet the end conditions. Generally, shooting methods are very sensitive to variations in the initial guess.

A second indirect method is a *gradient* method, the method of *steepest descent.* With gradient methods the dynamic system equations are solved exactly at each iteration and the control is slightly perturbed. The control history is adjusted at each step to further reduce the cost. The state equation (1.8)

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{1.11}$$

is integrated from t_0 to t_f to obtain $\mathbf{x}(t)$ for a guessed $\mathbf{u}(t)$, $t \in [t_0, t_f]$. The cost sensitivity matrices and Lagrangian gradients are then evaluated. The adjoint vector is integrated backwards to obtain $\boldsymbol{\lambda}(t)$, thus determining $\mathcal{H}_u$. $\mathcal{H}_u$ may not start close to zero, but approaches zero with each iteration. This method of convergence makes stopping criteria very important to this method.

Direct methods have been developed to avoid solving the optimality conditions. Instead of starting by solving for the optimality conditions, direct methods begin by discretizing the problem. The states, controls or both are discretized and the continuous problem changes into a discrete problem before the cost function is optimized. This reduces the optimal control problem to a parameter optimization problem that can be solved by Nonlinear Programming (NLP) solvers. The accuracy of the solution depends on the discretization producing a problem that accurately represents the continuous problem. With the current advances in NLP solvers, a properly discretized problem will result in solutions that are increasingly more accurate. In the 1960s, Newton's method was used to solve direct problems and the size of the problems was limited to $n = m = 10$, with n states and m controls. In the 1970s, the use of quasi-Newton methods and reduced gradient methods increased the size to $n = m \leq 100$. Current advances in computing with sparse matrices in Numerical Linear Algebra have allowed solving problems with $n = m = 10,000$ [Ref. 3]. These advances have resulted in continued interest and research in direct methods.

This thesis focuses on a direct method of solving optimal control problems. It relies on a Legendre-Gauss-Lobatto Pseudospectral (LGLP) algorithm which was first utilized in [Ref. 4] for solving problems in optimal control theory. This algorithm discretizes the problem and converts the problem into an NLP. Further developments by Professor Fahroo and Professor Ross [Ref. 5], implemented the algorithm and performed costate estimation in MATLAB.

The original research in this project involved creating a Graphical User Interface (GUI) in MATLAB that allows optimal control problems to be solved using the LGLP method without requiring a thorough understanding of the method. The end state of this project is a GUI that only requires understanding of the basic optimal control problem formulation to allow solution of a wide variety of problems. The LGLP code relies on codes developed by Professor William Gragg to solve Numerical Linear Algebra problems associated with the algorithm. In addition to a knowledge of MATLAB and Numerical Linear Algebra, a basic understanding of the Calculus of Variations and Optimal Control Theory is required for this thesis.

II. OPTIMAL CONTROL PROBLEMS

A dynamic system that represents an optimal control problem is mathematically described by a system of ordinary differential equations. The optimal control problem is defined by state equations, boundary conditions on the state variables, equality constraints and/or inequality constraints on the state variables, constraints on the controls, and a cost function to be optimized.

Definition: Let $s(\mathbf{y})$ be a scalar function of $\mathbf{y} = [y_1 \dots y_n]^T$. The gradient of s with respect to $\mathbf{y}$ is defined as the $n \times 1$ vector:

$$\frac{\partial s}{\partial \mathbf{y}}(\mathbf{y}) \equiv \begin{bmatrix} \frac{\partial s}{\partial y_1}(\mathbf{y}) \\ \frac{\partial s}{\partial y_2}(\mathbf{y}) \\ \vdots \\ \frac{\partial s}{\partial y_n}(\mathbf{y}) \end{bmatrix}. \tag{2.1}$$

Definition: If $\mathbf{a}(\mathbf{y})$ is an $n \times 1$ vector function of $\mathbf{y}$ (an $m \times 1$ vector), the Jacobian matrix is defined as the $n \times m$ matrix:

$$\frac{\partial [\mathbf{a}(\mathbf{y})]}{\partial \mathbf{y}} \equiv \begin{bmatrix} \frac{\partial a_1}{\partial \mathbf{y}_1}(\mathbf{y}) & \frac{\partial a_1}{\partial \mathbf{y}_2}(\mathbf{y}) & \dots & \frac{\partial a_1}{\partial \mathbf{y}_m}(\mathbf{y}) \\ \frac{\partial a_2}{\partial \mathbf{y}_1}(\mathbf{y}) & \frac{\partial a_2}{\partial \mathbf{y}_2}(\mathbf{y}) & \dots & \frac{\partial a_2}{\partial \mathbf{y}_m}(\mathbf{y}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial a_n}{\partial \mathbf{y}_1}(\mathbf{y}) & \frac{\partial a_n}{\partial \mathbf{y}_2}(\mathbf{y}) & \dots & \frac{\partial a_n}{\partial \mathbf{y}_m}(\mathbf{y}) \end{bmatrix}. \tag{2.2}$$

A. COST FUNCTIONS

The cost function to be optimized is a function of the states, controls and final time. There are three types of cost functions: Mayer, Lagrange and Bolza. A Mayer cost function is of the form:

$$\mathcal{J}(\mathbf{u}, \mathbf{x}, t_f) = M(\mathbf{x}, t_f). \tag{2.3}$$

The Lagrange type of cost function has an integral cost. It has the form:

$$\mathcal{J}(\mathbf{u}, \mathbf{x}, t_f) = \int_{t_0}^{t_f} \mathcal{L}(\mathbf{x}, \mathbf{u}) dt. \tag{2.4}$$

The Bolza cost function incorporates both the Mayer and Lagrange forms:

$$\mathcal{J}(\mathbf{u},\mathbf{x},t_f) = M(\mathbf{x},t_f) + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{x},\mathbf{u})dt. \tag{2.5}$$

It can be shown that the three types of cost functions are equivalent and any type of cost function may be transformed into either of the other types [Ref. 6].

B. STATE EQUATIONS AND CONSTRAINTS

The solution of the problem is found by determining the control function $\mathbf{u}(t)$, and the corresponding state trajectory $\mathbf{x}(t)$, that minimizes the cost function. The problem has a state vector $\mathbf{x} \in R^n$ and a control vector $\mathbf{u} \in R^m$. The dynamic equations for the states are expressed as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t),\mathbf{u}(t)), \qquad t \in [t_0, t_f] \tag{2.6}$$

with initial and final boundary conditions:

$$\boldsymbol{\psi}_0[\mathbf{x}(t_0), t_0] = 0, \tag{2.7}$$

$$\boldsymbol{\psi}_f[\mathbf{x}(t_f), t_f] = 0, \tag{2.8}$$

where $\boldsymbol{\psi}_0 \in R^p$ with $p \leq n$ and $\boldsymbol{\psi}_f \in R^q$ with $q \leq n$. The problem may have inequality constraints on the controls which are formulated as

$$\mathbf{g}[\mathbf{u}(t)] \leq 0 \tag{2.9}$$

where $\mathbf{g} \in R^r$ and $\frac{\partial \mathbf{g}}{\partial \mathbf{u}}$ has full rank.

C. THE ADJOINT DIFFERENTIAL EQUATIONS

Once the problem is mathematically formulated, the principles from calculus of variations are used to derive necessary and sufficient conditions for optimality. We adjoin the state equations and constraints to the cost function to create an augmented cost function. For simplicity in the presentation of the adjoint differential equations and the following optimality conditions, we make the simplifying assumption that

$$\mathbf{g}[\mathbf{u}(t)] = 0. \tag{2.10}$$

For a complete treatment of the inequality case, reference [Ref. 14]. The augmented cost function is

$$\mathcal{J}_A = \mathcal{J} + \boldsymbol{\lambda}^T\mathbf{f} + \boldsymbol{\mu}^T\mathbf{g} + \boldsymbol{\nu}_0^T\boldsymbol{\psi}_0 + \boldsymbol{\nu}_f^T\boldsymbol{\psi}_f. \tag{2.11}$$

Using Lagrange multiplier theory, $\boldsymbol{\lambda}(t) \in R^n$, $\boldsymbol{\mu}(t) \in R^r$, $\boldsymbol{\nu}_0(t) \in R^p$, with $p \le n$, and $\boldsymbol{\nu}_f(t) \in R^q$, with $q \le n$, are the Lagrange multipliers. The variable $\mathbf{f}$ represents the state equations, $\boldsymbol{\psi}_0$ represents initial time constraints, $\mathbf{g}$ represents control constraints, and $\boldsymbol{\psi}_f$ represents final time constraints. Since the state equation is

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \tag{2.12}$$

we adjoin the constraint

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) - \dot{\mathbf{x}} = 0 \tag{2.13}$$

to the cost function resulting in

$$\mathcal{J}_A = \mathcal{J} + \boldsymbol{\lambda}^T(\mathbf{f}(\mathbf{x}, \mathbf{u}) - \dot{\mathbf{x}}) + \boldsymbol{\mu}^T\mathbf{g}(\mathbf{u}) + \boldsymbol{\nu}_0^T\boldsymbol{\psi}_0 + \boldsymbol{\nu}_f^T\boldsymbol{\psi}_f \tag{2.14}$$

Using a Bolza cost function, the augmented cost function is

$$\mathcal{J}_A = M(\mathbf{x}, t_f) + \boldsymbol{\nu}_0^T\boldsymbol{\psi}_0 + \boldsymbol{\nu}_f^T\boldsymbol{\psi}_f + \int_{t_0}^{t_f} \left[\mathcal{L}(\mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^T\left(\mathbf{f}(\mathbf{x}, \mathbf{u}) - \dot{\mathbf{x}}\right) + \boldsymbol{\mu}^T\mathbf{g}(\mathbf{u})\right] dt. \tag{2.15}$$

Using the Hamiltonian in the cost function will simplify the equations. The Hamiltonian is defined as

$$\mathcal{H}(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}, \boldsymbol{\mu}) = \mathcal{L}(\mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^T\mathbf{f}(\mathbf{x}, \mathbf{u}) + \boldsymbol{\mu}^T\mathbf{g}(\mathbf{u}). \tag{2.16}$$

Thus by expanding the cost function,

$$\mathcal{J}_A = M(\mathbf{x}, t_f) + \boldsymbol{\nu}_0^T\boldsymbol{\psi}_0 + \boldsymbol{\nu}_f^T\boldsymbol{\psi}_f + \int_{t_0}^{t_f} \mathcal{L}(\mathbf{x}, \mathbf{u})\, dt + \int_{t_0}^{t_f} \boldsymbol{\mu}^T\mathbf{g}\, dt + \int_{t_0}^{t_f} \boldsymbol{\lambda}^T\mathbf{f}(\mathbf{x}, \mathbf{u})\, dt - \int_{t_0}^{t_f} \boldsymbol{\lambda}^T\dot{\mathbf{x}}\, dt, \tag{2.17}$$

and substituting the Hamiltonian, we have

$$\mathcal{J}_A = M(\mathbf{x}, t_f) + \boldsymbol{\nu}_0^T\boldsymbol{\psi}_0 + \boldsymbol{\nu}_f^T\boldsymbol{\psi}_f + \int_{t_0}^{t_f} \mathcal{H}(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}, \boldsymbol{\mu})\, dt - \int_{t_0}^{t_f} \boldsymbol{\lambda}^T\dot{\mathbf{x}}\, dt. \tag{2.18}$$

For simplicity, the augmented Bolza cost function is expressed as

$$\mathcal{J}_A = M[\cdot] + \boldsymbol{\nu}_0^T \psi_0 + \boldsymbol{\nu}_f^T \boldsymbol{\psi}_f + \int_{t_0}^{t_f} \mathcal{H}[\cdot]\, dt - \int_{t_0}^{t_f} \boldsymbol{\lambda}^T \dot{\mathbf{x}} dt. \tag{2.19}$$

Using integration by parts in the second integral we have

$$\int_{t_0}^{t_f} \boldsymbol{\lambda}^T \dot{\mathbf{x}}\, dt = \boldsymbol{\lambda}^T \mathbf{x}\Big|_{t_0}^{t_f} - \int_{t_0}^{t_f} \dot{\boldsymbol{\lambda}}^T \mathbf{x}\, dt. \tag{2.20}$$

This allows simplification of the cost function to

$$\mathcal{J} = M[\cdot] + \boldsymbol{\nu}_0^T \psi_0 + \boldsymbol{\nu}_f^T \boldsymbol{\psi}_f - \boldsymbol{\lambda}^T \mathbf{x}\Big|_{t_f} + \boldsymbol{\lambda}^T \mathbf{x}\Big|_{t_0} + \int_{t_0}^{t_f} \mathcal{H}[\cdot] dt + \int_{t_0}^{t_f} \dot{\boldsymbol{\lambda}}^T \mathbf{x}\, dt. \tag{2.21}$$

D. NECESSARY OPTIMALITY CONDITIONS

The minimum of the functional, $\mathcal{J}$, for the optimal control $\mathbf{u}^*$ occurs when

$$\mathcal{J}(\mathbf{u}) - \mathcal{J}(\mathbf{u}^*) = \Delta\mathcal{J} \geq 0 \tag{2.22}$$

for all controls $\mathbf{u}$ close to $\mathbf{u}^*$. Let $\mathbf{u} = \mathbf{u}^* + \delta\mathbf{u}$ and

$$\Delta\mathcal{J}(\mathbf{u}^*, \delta\mathbf{u}) = \delta\mathcal{J}(\mathbf{u}^*, \delta\mathbf{u}) + \text{ higher order terms.} \tag{2.23}$$

$\delta\mathcal{J}$ is linear in $\delta\mathbf{u}$ and the higher-order terms approach zero as the norm of $\delta\mathbf{u}$ approaches zero. The increment of J is

$$\begin{aligned}
\Delta\mathcal{J}(\mathbf{u}^*, \delta\mathbf{u}) \;=\; & \left[\frac{\partial M}{\partial \mathbf{x}_f}(\mathbf{x}(t_f), t_f) + \left(\frac{\partial \boldsymbol{\psi}_f}{\partial \mathbf{x}_f}\right)^T \boldsymbol{\nu}_f - \boldsymbol{\lambda}(t_f)\right]^T \delta\mathbf{x}_f \\
& + \left[\mathcal{H}(\mathbf{x}(t_f), \mathbf{u}(t_f), \boldsymbol{\lambda}(t_f), \boldsymbol{\mu}(t_f), t_f) + \left(\frac{\partial \boldsymbol{\psi}_f}{\partial t_f}\right)^T \boldsymbol{\nu}_f + \frac{\partial M}{\partial t_f}(\mathbf{x}(t_f), t_f)\right] \delta t_f \\
& + \int_{t_0}^{t_f} \left\{ \left[\dot{\boldsymbol{\lambda}}(t) + \frac{\partial \mathcal{H}}{\partial \mathbf{x}}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t)\right]^T \delta\mathbf{x}(t) \right. \\
& + \left[\frac{\partial \mathcal{H}}{\partial \mathbf{u}}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t)\right]^T \delta\mathbf{u}(t) \\
& \left. + \left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\mu}}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t)\right]^T \delta\boldsymbol{\mu}(t) \right\} dt \\
& + \text{higher order terms.}
\end{aligned} \tag{2.24}$$

By setting the coefficients of the differentials equal to zero, various state and control constraints are obtained.[1] If the state equations are satisfied, and the constraints on control are satisfied

$$\left[\frac{\partial \mathcal{H}}{\partial \boldsymbol{\mu}}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t)\right] = 0, \tag{2.25}$$

then $\boldsymbol{\lambda}(t)$ is selected so that

$$\left[\dot{\boldsymbol{\lambda}}(t) + \frac{\partial \mathcal{H}}{\partial \mathbf{x}}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t)\right] = 0, \tag{2.26}$$

and the final time conditions are

$$\begin{aligned} & \left[\frac{\partial M}{\partial \mathbf{x}}(\mathbf{x}(t_f), t_f) + \left(\frac{\partial \psi_f}{\partial \mathbf{x}_f}\right)^T \boldsymbol{\nu}_f - \boldsymbol{\lambda}(t_f)\right]^T \delta \mathbf{x}_f \qquad (2.27) \\ + & \left[\mathcal{H}(\mathbf{x}(t_f), \mathbf{u}(t_f), \boldsymbol{\lambda}(t_f), \boldsymbol{\mu}(t_f), t_f) + \left(\frac{\partial \psi_f}{\partial t_f}\right)^T \boldsymbol{\nu}_f + \frac{\partial M}{\partial t_f}(\mathbf{x}(t_f), t_f)\right] \delta t_f \quad = \quad 0. \end{aligned}$$

For final state specified and final time free, $\delta \mathbf{x}_f = 0$ and δt_f is free. For final state free and final time specified, $\delta \mathbf{x}_f$ is free and $\delta t_f = 0$. For more complicated examples refer to [Ref. 2].

The increment of $\mathcal{J}$ simplifies to

$$\Delta \mathcal{J}(\mathbf{u}^*, \delta \mathbf{u}) = \int_{t_0}^{t_f} \left\{ \left[\frac{\partial \mathcal{H}}{\partial \mathbf{u}}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t)\right]^T \delta \mathbf{u}(t) \right\} dt + \text{higher order terms.} \tag{2.28}$$

This integral is the first order approximation of the change in $\mathcal{H}$ due to the control $\mathbf{u}$,

$$\begin{aligned} \left[\frac{\partial \mathcal{H}}{\partial \mathbf{u}}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t)\right] \quad \approx \quad & \mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t) + \delta \mathbf{u}, \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t) \\ & - \mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t). \end{aligned} \tag{2.29}$$

Simplifying, the increment of $\mathcal{J}$ is

$$\begin{aligned} \Delta \mathcal{J}(u^*, \delta u) \quad = \quad & \int_{t_0}^{t_f} \{\mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t) + \delta \mathbf{u}, \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t) \\ & - [\mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t)]\} \; dt + \text{higher order terms.} \end{aligned} \tag{2.30}$$

---

[1] In our notation the gradient of a scalar is a $n \times 1$ vector and the Jacobian of an $n \times 1$ vector of m variables has dimension $n \times m$. The variational $\delta \mathbf{x}$ is also $n \times 1$.

If $\mathbf{u}^* + \delta\mathbf{u}$ is in a suitably small neighborhood of $\mathbf{u}^*$, ($\|\delta\mathbf{u}\| < \beta$), the higher order terms may be neglected and the necessary condition for optimality is

$$\mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t) + \delta\mathbf{u}, \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t) - \mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t) \geq 0 \tag{2.31}$$

or

$$\mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t) + \delta\mathbf{u}, \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t) \geq \mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t). \tag{2.32}$$

The necessary condition

$$\mathcal{H}(\mathbf{x}(t), \mathbf{u}^*(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t) \leq \mathcal{H}(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), \boldsymbol{\mu}(t), t) \tag{2.33}$$

is Pontryagin's minimum principle. The minimal principle states that an optimal control must minimize the Hamiltonian. [Ref. 2]

In summary, the optimal control problem is formulated as follows: find controls, $\mathbf{u}^* \in U$, the set of all possible controls, that cause the system described by Equations (2.6 -2.8) and (4.7) to follow a feasible trajectory to minimize the cost function, Equation (2.5). The necessary conditions for optimality are Equations (2.25-2.28) and Pontryagin's minimum principle Equation (2.33).

E. NUMERICAL SOLUTIONS

Once the necessary conditions have been derived, an indirect numerical method may be used to solve the resulting system. The derivation of the problem is essential to the understanding of the problem, but is only required in the indirect solution techniques. Solving the nonlinear BVP problem that is described by this system of equations can be very difficult. To avoid this computational difficulty, we will focus on direct methods. The use of a direct method allows the solution of the optimal control problem by use of existing NLP solvers and does not require solving for the optimality conditions.

III. DIRECT METHODS

Direct methods are more recent developments when compared with their indirect method counterparts. All direct methods involve deciding upon a finite set of variables and then iterating by Newton's method or another zero finding algorithm to numerically solve the resulting problem. The essential difference between direct and indirect methods lies in the direct methods converting the optimal control problem into an NLP and directly solving the NLP, instead of deriving the optimality conditions and solving the resulting BVP. The two most commonly used direct methods are *direct shooting* and *direct transcription* algorithms [Ref. 3]. *Differential inclusion*, a technique for reducing the size of the NLP is also included in this thesis.

In direct methods, the Karush-Kuhn-Tucker conditions from NLP theory are solved during the optimization instead of solving for the optimal control necessary conditions. It is not surprising that when the discrete conditions are taken to the limit, determining the NLP active set is equivalent to finding the constrained subarcs and junction points in the continuous optimal control problem.

The first step in a direct method is to select a method to discretize the problem and convert it into a parameter optimization problem. The problem is discretized by dividing the time interval into subintervals whose endpoints are called nodes. These nodes will define the variables for the NLP problem. The unknowns are the values of the controls, states, and problem parameters at the nodes. These variables are the state and control parameters for the new parameter optimization problem. When the state equations and cost function are expressed in terms of these parameters, the optimal control problem is reduced to an NLP. [Ref. 7]

The second selection that defines a direct method is the choice of an interpolation scheme to interpolate the values or time histories of the control and state variables between the discrete nodes. Either an explicit or implicit scheme will be chosen to describe the equations of motion. An explicit scheme solves the ordinary differential equations (ODE) to develop the time histories between the nodes. An implicit scheme uses integration rules to create formulas for the integrals and then adds these equations as constraints to the NLP. Many implicit interpolation schemes use orthogonal polynomials to interpolate the state and control values between the nodes. One advantage of using orthogonal polynomials is their close relationship to Gauss integration rules, which results in highly accurate quadrature rules.

For explicit integration the value of $\mathbf{x}$, the state variable, is needed to evaluate the value of the function at each discretization point. Explicit integration allows integration of the state equations from initial to final time in one pass. With implicit integration, the values of $\mathbf{x}$ are not known in advance and as a result a predictor-corrector approach must be used.

It is desirable for all integration rules to be of the highest order possible. The higher the order of an integration scheme, the smaller the error from a single integration step taken with a step size δ. For example, with a step size of $\delta = 0.1$, and using Euler's quadrature rule whose local error is $O(\delta^2)$, the error would be ≈ 0.01. For a higher order rule, such as Simpson's rule with a local error of $O(\delta^5)$, the error would be ≈ 0.00001 for the same step size. This error dependence on step size and integration rule is shared by all the direct methods.

Inequality constraints make optimal control problems increasingly difficult and efficiency of handling such constraints is very important. Constrained arcs are often not known a priori and the junction points from constrained to unconstrained arcs may also not be known. These discontinuities of the control and adjoint variables at the junction points are essentially boundary conditions and transform the problem into a multi-point BVP.

Table I illustrates a basic algorithm for a direct method to convert an optimal control problem into a parameter optimization problem, equivalent to an NLP [Ref. 7]. The direct methods mentioned here, direct shooting and direct collocation, both differ in their choice of variables for the NLP. The direct shooting method discretizes

1. Divide the time interval.
2. Choose variables to be calculated by interpolation.
3. Integrate the state equations explicitly or implicitly.
4. Solve the NLP.

Table I. Direct Method Algorithm

the control history, and direct collocation discretizes both states and controls. This difference tends to make a direct collocation NLP larger than the resulting systems from the other methods. The following summaries of direct methods are taken from John Betts "Survey of Numerical Methods for Trajectory Optimization" [Ref. 3].

A. DIRECT SHOOTING

In a direct shooting method the variables for the NLP are chosen from the initial conditions, final conditions and problem parameters. Phases are chosen at points where the problem is discontinuous and at boundaries between constrained and unconstrained arcs. The control history will be represented by a finite set of parameters. For each phase, k, the NLP variables are defined as

$$X^{(k)} = \{\mathbf{x}(t_0), \mathbf{x}(t_f), t_0, t_f, \mathbf{p}\} \tag{3.1}$$

where the $\mathbf{x}$ represents the state equations and $\mathbf{p}$ represents a set of parameters. [Ref. 7]

The control $\mathbf{u}$ can be explicitly or implicitly represented by $\mathbf{p}$ as in the following examples

$$u = p_1 + p_2 t \tag{3.2}$$

or implicitly as

$$0 = p_1 u(t) + \sin[p_2 u(t)]. \tag{3.3}$$

The total set of NLP variables will be

$$x \in \{X^{(1)}, X^{(2)}, \ldots, X^{(N)}\}. \tag{3.4}$$

Solving for the constraints at the end points of each phase provides constraints

$$g(x) = \left\{\boldsymbol{\psi}^{(1)}(\mathbf{x}(t_0), t_0, \mathbf{p}), \boldsymbol{\psi}^{(2)}(\mathbf{x}(t_1), t_1, \mathbf{p}), \ldots, \boldsymbol{\psi}^{(N)}(\mathbf{x}(t_f), t_f, \mathbf{p})\right\} \tag{3.5}$$

where $\boldsymbol{\psi}^{(k)}$ denotes a function that describes the boundary conditions. [Ref. 3]

The Program to Optimize Simulated Trajectories (POST) developed by Martin Marietta and the Generalized Trajectory Simulation (GST) program by the Aerospace Corporation are both implementations of this method. The flight avionics on the space shuttle also incorporate a direct shooting algorithm for steering. [Ref. 3]

The direct shooting algorithm has advantages for launch vehicle and orbit transfer problems. These types of problems result in an NLP represented by a small number of variables. The direct shooting algorithm with a large number of variables tends to lose accuracy due to propagation of early errors throughout the procedure. Another disadvantage is the cost of computing the gradient functions. The system equations must be integrated forward numerically in direct shooting methods. Normally, finite difference codes are used to solve for the gradients. In addition to the numeric complexity involved with solving for the gradient of a large system, there are also problems with the accuracy of the gradient information. Using forward difference methods to calculate the gradients involves an error of the order of the step size, $O(\delta)$. Using central difference methods to calculate the gradients is twice as expensive as forward difference methods, but the error is $O(\delta^2)$.

A direct shooting algorithm is contained in Table II. Within Table II, $\mathbf{x}$ represents the state equations, $\mathbf{u}$ represents the control, and $\boldsymbol{\psi}$ describes the boundary conditions.

Direct shooting methods are very efficient when a detailed mathematical model is not required. In many modern problems, such as problems using current rockets which do not have variable rate thrusters and have a very high weight to force ratio,

1. Divide the problem into k number of phases based upon arc constraints and inequalities.
2. Generate the control $\mathbf{u}$ implicitly or explicitly from a finite set of parameters $\mathbf{p}$.
3. Compute initial conditions, $\boldsymbol{\psi}^k[\mathbf{x}^k(t_0), \mathbf{p}^k, t_0]$, for each phase numerically.
4. Solve initial value problem for the subarcs.
5. Compute final conditions, $\boldsymbol{\psi}^k[\mathbf{x}^k(t_f), \mathbf{p}^k, t_f]$, for each phase numerically.
6. Solve the resulting NLP.

Table II. Direct Shooting Algorithm

detailed mathematical models are unnecessary [Ref. 3]. The small number of variables in the NLPs developed by the direct shooting method speed up convergence to a solution. When a large time interval is used, the conversion from a single integration step to multiple integration steps to cover the time interval converts the standard direct shooting method to a multiple direct shooting method.

B. DIRECT COLLOCATION

A direct collocation method discretizes both the control and the state equations. The values will be known exactly at the nodes of the discrete time intervals. The integration is completed by defining the residuals for each integration step and then driving the residuals to zero during the solution of the NLP [Ref. 7]. The NLP variables are exactly the values of the state and control equations evaluated at the collocation points. When low order rules are used, the method is called transcription. When higher order rules are used, the method is termed collocation. Euler and trapezoidal rules are used with transcription methods while Simpson's rule, Gauss-Lobatto, and other higher order rules are used in collocation methods.

A direct collocation method's strength lies in not requiring prior knowledge or specification of the arc sequence for path inequalities. Singular arcs arise when the control derived is not uniquely defined by the optimal conditions. Whenever the second partial of the Hamiltonian with respect to the control is equal to zero, $\mathcal{H}_{uu} = 0$, the optimal solution computed will not be unique.

The direct collocation method results in very large NLPs, but the Jacobian and the Hessian matrices that result are sparse, allowing for efficient codes from recent advancements in numerical linear algebra to exploit these matrix structures. It is not uncommon for the matrices involved in the solution of a direct collocation problem to have only one percent non-zero entries.

In addition to the efficient codes used with collocation methods, the high order quadrature rules used in the implicit integration of the state equations allow for a larger step size with an equally high degree of accuracy when compared with the other methods listed here. Using Simpson's rule the local error is $O(\delta^5)$ and with Gauss Lobatto the local error is $O(\delta^8)$.

The implicit integration rules are added to the NLP as nonlinear constraint equations. The piecewise smooth interpolating polynomials used with direct collocation will satisfy the ODEs exactly at the collocation points of each interval. With collocation, the states are unknown and therefore the equations of motion must be integrated implicitly at each node, with each node representing one integration step. The size of the step dictates the number of integration steps required. A balance is desired between a small step size and increasing the number of integrations. A direct collocation algorithm is shown in Table III. [Ref. 3]

Direct collocation methods have been used to efficiently solve low thrust orbit transfer, commercial aircraft mission analysis, chemical process control and robotics problems. The Optimal Trajectories by Implicit Simulation (OTIS) program was developed for NASA and the U. S. Air Force, and has been widely distributed [Ref. 3]. OTIS uses a direct collocation method with Simpson's quadrature rule and Hermite cubic polynomial approximation of the state equations [Ref. 8]. The OTIS library implements a sparse nonlinear programming algorithm using NPSOL as the NLP solver.

1. Select number of discretization points, k.
2. Evaluate constraints at each node.
3. Calculate the integral residual for each step.
4. Compute initial conditions ψ_0.
5. Solve the resulting NLP.

Table III. Direct Collocation Algorithm

In addition, the Advanced Launch Trajectory Optimization Software (ALTOS) program developed for the European Space Agency uses many of the same features as OTIS.

C. DIFFERENTIAL INCLUSION

Differential inclusion is very effective on problems that have linear controls. Like collocation, differential inclusion uses implicit integration rules to formulate nonlinear constraint equations that are used in the NLP formulation [Ref. 8]. Unlike collocation, however, differential inclusion has only been successfully implemented with Euler's method. The linear controls allow the controls to be represented explicitly in terms of the states and their rate of change (derivative). The method eliminates the controls from the system of equations by explicitly solving for the controls, and then discretizing only the state equations to reduce the number of NLP variables. This is effective only when an explicit formula for the controls in terms of the states can be found. Since in NLP problems, the central processing unit (CPU) usage increases geometrically with the number of variables, eliminating the controls simplifies the problem and speeds convergence to a solution. This reduced problem size is the advantage to differential inclusion.

Low order quadrature rules are used to integrate the state equations. Euler's rule, the most common quadrature rule that has been implemented with differential inclusion, has local error $O(\delta^2)$. Low order rules are required to effectively isolate the state and system variables at each node as a function of only the discrete states. The low order of the integration rules used in this method require selection of an increasingly smaller step size to increase accuracy. While decreasing the local error, decreasing the step size also increases the total number of integration steps required, as one step is required for each node.

Another disadvantage is that if the control histories are required, they must be solved for after the optimization. Differential inclusion also requires gradient information during the optimization process and requires prior knowledge of arc inequalities. Prior knowledge of the arc inequalities limits the robustness of this approach and calculating the gradients analytically for differential inclusion can be complicated [Ref. 8].

An algorithm using differential inclusion is shown in Table IV [Ref. 7]. One effective use of differential inclusion is to attempt to solve a problem for which collocation has failed to converge. The decreased problem size may allow the new NLP to converge [Ref. 8].

1. Solve for controls explicitly in terms of states and their derivatives.
2. Eliminate control variables from the system.
3. Discretize the resulting constraints and inequalities at the nodes.
4. Compute initial conditions, ψ_0.
5. Solve the resulting NLP.
6. Solve for the resulting control histories (if required).

Table IV. An Algorithm using Differential Inclusion

D. SUMMARY OF DIRECT METHODS

Each of the direct methods contain a mix of advantages and disadvantages based upon their discretization and solution methodology. All direct methods share the advantage of not requiring the solution of, or solving for the optimality conditions. Direct shooting and differential inclusion both tend to result in smaller NLPs than those derived through collocation. Direct Shooting is very efficient when simple mathematical models suffice to control the system. Differential inclusion works very well when the controls are linear and can be solved for explicitly in terms of the state equations and their rate of change.

Collocation effectively takes advantage of high order integration rules. The advantages of high order integration rules allowing a larger step size with high accuracy make collocation very robust and effective. The combination of the accuracy of discretizing both the states and controls, and the higher order rules available, often outweigh the larger NLP formulations, especially for complicated systems. We chose to use a collocation method in this thesis.

IV. A PSEUDOSPECTRAL METHOD

The Legendre-Gauss-Lobatto Pseudospectral method (LGLP) is a direct collocation method that incorporates spectral methods adapted from work in Computational Fluid Dynamics (CFD) [Ref. 9]. LGLP is a direct collocation method that uses a high order integration rule, Legendre-Gauss-Lobatto, and discretizes the time histories for both the states and controls. The use a of high order integration rule helps to limit the number of variables required for the NLP to achieve sufficient accuracy by allowing a large step size.

Pseudospectral implies that the method has spectral accuracy. Spectral accuracy refers to the property of orthogonal series, especially Fourier Series, that the kth coefficient of the expansion decays faster than any inverse power of k when the function is smooth and the derivatives are periodic. This gives the Fourier Series, and other orthogonal expansions, the property that the truncated series, with a few more terms, gives an exceedingly good approximation of the function, assuming proper smoothness of the function. [Ref. 9]

We use orthogonal Legendre polynomials to approximate the state and control variables. One advantage of using orthogonal polynomials is their close relationship to Legendre-Gauss-Lobatto integration rules. This is exploited with the integral portion of the cost function and in the implicit integration of the state equations to transform the optimal control problem into a system of algebraic equations. Polynomial approximations of the state and control variables are used where Lagrange polynomials are the trial functions and the coefficients are the values of the state and control variables at the Legendre-Gauss-Lobatto (LGL) points. The cost function is discretized using Legendre-Gauss-Lobatto quadrature rules.

A. DISCRETIZING THE PROBLEM

The LGLP method can be used for solving optimal control problems formulated as in Chapter II. The polynomial approximations for the state and control functions are calculated in terms of their values at the LGL points lying in the interval $[-1, 1]$. We use a transformation to express the problem for $\tau \in [-1, 1]$:

$$t = \frac{(t_f - t_0)\tau + (t_f + t_0)}{2} \tag{4.1}$$

which is equivalent to

$$\tau = \frac{2t - (t_f + t_0)}{t_f - t_0}. \tag{4.2}$$

As a result of the transformation $t \to \tau$ we must adopt new symbols for the variables

$$\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}$$

and the maps

$$\mathcal{J}(\cdot), \mathcal{L}(\cdot), \mathcal{M}(\cdot), \mathbf{f}(\cdot), \boldsymbol{\psi}_0(\cdot), \boldsymbol{\psi}_f(\cdot).$$

Using the following changes of variables

$$\mathbf{x} \to \mathbf{y}, \mathbf{u} \to \mathbf{v}, \boldsymbol{\lambda} \to \boldsymbol{\Lambda}, \boldsymbol{\mu} \to \boldsymbol{\zeta}, \boldsymbol{\nu} \to \boldsymbol{\eta},$$

and the following change of mappings

$$\mathcal{J}(t) \to \mathbf{J}(\tau), \mathcal{L}(t) \to \mathbf{L}(\tau), M(t) \to \mathcal{M}(\tau)$$

$$\mathbf{f}(t) \to \mathcal{F}(\tau), \boldsymbol{\psi}_0(t) \to \boldsymbol{\Psi}_0(\tau), \boldsymbol{\psi}_f(t) \to \boldsymbol{\Psi}_f(\tau).$$

It follows that by using Equation (4.1), Equations (2.5-2.9) and (2.16)can be replaced by

$$\begin{aligned}
\mathbf{J}(\mathbf{y}(\cdot), \mathbf{v}(\cdot), t_f) &= \mathcal{M}[\mathbf{y}(1), t_f] + \frac{t_f - t_0}{2}\int_{-1}^{1} \mathbf{L}[\mathbf{y}(\tau), \mathbf{v}(\tau)]d\tau & (4.3)\\
\dot{\mathbf{y}}(\tau) &= \left(\frac{t_f - t_0}{2}\right)[\mathcal{F}(\mathbf{y}(\tau), \mathbf{v}(\tau))], & (4.4)\\
\boldsymbol{\Psi}_0(\mathbf{y}(-1), t_0) &= 0 & (4.5)\\
\boldsymbol{\Psi}_f(\mathbf{y}(1), t_f) &= 0 & (4.6)\\
\mathbf{g}(\mathbf{v}(\tau)) &\leq \mathbf{0} & (4.7)\\
\mathcal{H}(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{v}) &= \left(\frac{t_f - t_0}{2}\right)\boldsymbol{\Lambda}^T\mathcal{F} + \left(\frac{t_f - t_0}{2}\right)\mathbf{L} + \boldsymbol{\zeta}^T\mathbf{g} & (4.8)
\end{aligned}$$

B. LEGENDRE POLYNOMIALS

There are three main classes of classic orthogonal polynomials, Jacobi, Laguere and Hermite polynomials [Ref. 10]. Legendre and Chebyshev polynomials, both used in collocation schemes for direct methods, are of the Jacobi polynomials class. One advantage of using orthogonal polynomials is that their derivatives also form a set of orthogonal polynomials [Ref. 10]. The polynomials are bounded in every proper subinterval of $(-1, 1)$, which allows a transformation to $L_2[-1, 1]$, to ensure the states and controls will be bounded.

The Rodriguez formula is commonly used to express Jacobi polynomials

$$(1-x)^{\beta}(1+x)^{\gamma}P_n^{(\beta,\gamma)}(x) = \frac{(-1)^n}{2^n n!}\frac{d^n}{dx^n}\left[(1-x)^{n+\beta}(1+x)^{n+\gamma}\right] \text{[Ref. 10]}. \tag{4.9}$$

$P_n(x)$ represents the orthogonal polynomial of the nth degree evaluated at the point x. The variables β and γ are parameters which define the type of polynomial defined by the Rodriguez formula. When $\beta, \gamma > -1$, the polynomials expressed by the formula are Jacobi polynomials. The weight function that characterizes the orthogonal polynomial, $(1-x)^{\beta}(1+x)^{\gamma}$, is also incorporated in Equation (4.9). The Legendre Polynomials are defined by $\beta = \gamma = 0$ resulting in a weight function $w(x) = 1$ [Ref. 11]. The Rodriguez formula for Legendre polynomials simplifies to

$$L_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n. \tag{4.10}$$

Thus, $L_n(x)$ is the Legendre polynomial of degree n with leading term $\frac{1}{2^n}\binom{2n}{n}$. The Legendre Polynomials may be generated by the three term recurrence relationship

$$L_{k+1}(x) = \frac{2k+1}{k+1}xL_k(x) - \frac{k}{k+1}L_{k-1}(x) \tag{4.11}$$

with

$$L_0(x) = 1 \text{ and } L_1(x) = x.$$

The orthogonal polynomials have a close relationship with the theory of real tridiagonal matrices. This relationship allows new numerical linear algebra codes that exploit the matrix structure to find the zeros of the Legendre polynomials. The roots χ_i , $i = 1, 2, \ldots, n$ are the eigenvalues of the tridiagonal matrix

$$J_n = \begin{bmatrix} \delta_1 & \gamma_2 & & \\ \gamma_2 & \delta_2 & \ddots & \\ & \ddots & \ddots & \gamma_n \\ & & \gamma_n & \delta_n \end{bmatrix}. \tag{4.12}$$

The unique orthogonal properties of the Legendre polynomials allow us to exploit their boundedness and structure by transforming the original problem space into $[-1, 1]$. The Legendre polynomials satisfy the ODE

$$(1 - x^2)\ddot{L}_n - 2x\dot{L}_n + n(n+1)y = 0 \qquad \text{[Ref. 11]}. \tag{4.13}$$

With Legendre polynomials we have the properties that

$$\int_{-1}^{1} L_m(x)L_n(x)dx = 0 \qquad m \neq n \tag{4.14}$$

$$\int_{-1}^{1} L_m(x)L_n(x)dx = \frac{2}{2n+1} \qquad m = n. \tag{4.15}$$

This allows us to neglect cross terms in the expansions of the state and control variables. When a function is expanded with Legendre polynomials, the approximation formulas are

$$f(x) = \sum_{n=0}^{\infty} a_n L_n(x) \tag{4.16}$$

and

$$a_n = \frac{2n+1}{2} \int_{-1}^{1} f(x)L_n(x)dx. \tag{4.17}$$

C. APPROXIMATING THE STATES AND CONTROLS

Let $L_N(\tau)$ be the Legendre polynomial of degree N on the interval $[-1, 1]$. In the Legendre collocation approximation of (4.3)-(4.8), we use the LGL points, $\tau_l, l = 0, 1, \ldots, N$ which are given by

$$\tau_0 = -1$$

$$\tau_1, \tau_2, \ldots \tau_{N-1},$$

the zeros of $\dot{L}_N$, the derivative of the Legendre polynomial L_N, and

$$\tau_N = 1.$$

For approximating the continuous equations, we seek polynomial approximations for the state and control equations. We define

$$\mathbf{y}^N(\tau) = \sum_{l=0}^{N} \mathbf{y}(\tau_l)\phi_l(\tau), \tag{4.18}$$

$$\mathbf{v}^N(\tau) = \sum_{l=0}^{N} \mathbf{v}(\tau_l)\phi_l(\tau), \tag{4.19}$$

where, for $l = 0, 1, ..., N$,

$$\phi_l(\tau) = \frac{1}{N(N+1)L_N(\tau_l)} \frac{(\tau^2-1)\dot{L}_N(\tau)}{\tau - \tau_l}, \tag{4.20}$$

are the Lagrange polynomials of order N. Notice that the Lagrange polynomials require our previous calculation of L_n and $\dot{L}_n$. It can be shown that

$$\phi_l(\tau_k) = \delta_{lk} = \begin{cases} 1 & \text{if } l = k \\ 0 & \text{if } l \neq k. \end{cases}$$

From this orthogonal property of ϕ_l it follows that

$$\mathbf{y}^N(\tau_l) = \mathbf{y}(\tau_l) \tag{4.21}$$

$$\mathbf{v}^N(\tau_l) = \mathbf{v}(\tau_l). \tag{4.22}$$

Thus the value of the interpolating polynomial at the LGL points is exactly the value of the continuous state and control functions evaluated at the LGL points.

To facilitate the NLP formulation, we use the notation

$$\mathbf{a}_l := \mathbf{y}(\tau_l),\ \mathbf{b}_l := \mathbf{v}(\tau_l),$$

to rewrite (4.18)-(4.19) in the form:

$$\mathbf{y}^N(\tau) = \sum_{l=0}^{N} \mathbf{a}_l\phi_l(\tau), \tag{4.23}$$

$$\mathbf{v}^N(\tau) = \sum_{l=0}^{N} \mathbf{b}_l\phi_l(\tau). \tag{4.24}$$

D. CALCULATING DERIVATIVES

To express the derivative $\dot{\mathbf{y}}^N(\tau)$ in terms of $\mathbf{y}^N(\tau)$ at the collocation points τ_l, we differentiate (4.18). This differentiation of the approximating polynomial is accomplished by a matrix multiplication of the following form:

$$\dot{\mathbf{y}}^N(\tau_k) = \sum_{l=0}^{N} D_{kl}\mathbf{y}(\tau_l), \tag{4.25}$$

where D_{kl} are entries of the $(N+1) \times (N+1)$ differentiation matrix $\mathbf{D}$

$$\mathbf{D} := [D_{kl}] := \begin{cases} \frac{L_N(\tau_k)}{L_N(\tau_l)} \cdot \frac{1}{\tau_k - \tau_l} & k \neq l \\ -\frac{N(N+1)}{4} & k = l = 0 \\ \frac{N(N+1)}{4} & k = l = N \\ 0 & \text{otherwise} \end{cases} . \tag{4.26}$$

An example of the Legendre polynomial and the differentiation matrix for a fixed N is illustrative of this technique. Letting $N = 4$, we have the resulting 4th order Legendre polynomial,

$$L_4 = \frac{1}{8}(35x^4 - 30x^2 + 3). \tag{4.27}$$

Dividing the interval into four time periods and solving for the Lobatto points, the zeros of L_4, results in $\tau_0 = -1$, $\tau_1 = -.6547$, $\tau_2 = 0$, $\tau_3 = 0.6547$, $\tau_4 = 1$. The differentiation matrix results in a matrix of the following form

$$\begin{bmatrix} -5 & 6.7565 & -2.6667 & 1.4102 & -0.5 \\ -1.2410 & 0 & 1.7457 & -0.7638 & .2590 \\ 0.3750 & -1.3366 & 0 & 1.3366 & -0.3750 \\ -0.2590 & 0.7638 & -1.7457 & 0 & 1.2410 \\ 0.5 & -1.4102 & 2.6667 & -6.7565 & 5 \end{bmatrix} . \tag{4.28}$$

For the derivative of the state vector $\mathbf{y}(\tau)$, collocated at the points τ_k, we rewrite (4.25)

$$\mathbf{c}_k = \dot{\mathbf{y}}^N(\tau_k) = \sum_{l=0}^{N} D_{kl}\mathbf{a}_l. \tag{4.29}$$

E. DISCRETIZING INTEGRALS

Next, the integral (4.3) is discretized. Substituting (4.23) and (4.24) in (4.3) and using the Gauss-Lobatto integration rule, we obtain

$$\begin{aligned}
\mathbf{J}^N(\mathbf{a},\mathbf{b},t_f) &= \mathcal{M}(\mathbf{y}^N(1),t_f) + \frac{t_f - t_0}{2}\int_{-1}^{1}\mathbf{L}(\mathbf{y}^N,\mathbf{v}^N)d\tau \\
&= \mathcal{M}(\mathbf{y}^N(1),t_f) + \frac{t_f - t_0}{2}\sum_{k=0}^{N}\mathbf{L}\left(\sum_{l=0}^{N}\mathbf{a}_l\phi_l(\tau_k), \sum_{l=0}^{N}\mathbf{b}_l\phi_l(\tau_k)\right)w_k \\
&= \mathcal{M}(\mathbf{a}_N,t_f) + \frac{t_f - t_0}{2}\sum_{k=0}^{N}\mathbf{L}(\mathbf{a}_k,\mathbf{b}_k)w_k
\end{aligned} \tag{4.30}$$

The last equality is obtained from $\phi_l(t_k) = \delta_{lk}$. The coefficients are $\mathbf{a} = (\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_N)$,and $\mathbf{b} = (\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_N)$. The weights are given by

$$w_k := \frac{2}{N(N+1)}\frac{1}{[L_N(\tau_k)]^2} \quad k = 0, 1, \ldots, N.$$

F. THE PARAMETRIC OPTIMIZATION PROBLEM

The state equations and the initial and terminal state conditions are discretized by first substituting (4.23)-(4.24) in (4.8) and collocating at the LGL points, τ_k. Using the notation for $\mathbf{a}$ and $\mathbf{b}$, the state equations are transformed into the following algebraic equations

$$\mathbf{A}_k(\mathbf{a},\mathbf{b}) = \frac{t_f - t_0}{2}\mathcal{F}(\mathbf{a}_k,\mathbf{b}_k) - \mathbf{c}_k = 0, \qquad k = 0, 1, \ldots, N, \tag{4.31}$$

where $\mathbf{c}_k$ is as defined in (4.29), and the initial conditions are

$$\boldsymbol{\Psi}_0(\mathbf{y}^N(-1), t_0) = 0 \quad \text{or} \tag{4.32}$$

$$\boldsymbol{\Psi}_0(\mathbf{a}_0, t_0) = 0. \tag{4.33}$$

The terminal state conditions are

$$\boldsymbol{\Psi}_f(\mathbf{y}^N(1), t_f) = 0 \quad \text{or} \tag{4.34}$$

$$\boldsymbol{\Psi}_f(\mathbf{a}_N, t_f) = 0 \tag{4.35}$$

The control inequality constraints are approximated by

$$\mathbf{g}(\mathbf{v}^N(\tau_k)) \le 0,\ k = 0, 1, \ldots, N, \quad \text{or} \tag{4.36}$$

$$\mathbf{g}(\mathbf{b}_k) \le 0,\ k = 0, 1, \ldots, N. \tag{4.37}$$

To summarize, the optimal control problem (4.3)-(4.8) is approximated by the following nonlinear optimization problem: Find the coefficients

$$\mathbf{a} = (\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_N) \tag{4.38}$$

$$\mathbf{b} = (\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_N) \tag{4.39}$$

and possibly the final time t_f, to minimize the cost function

$$\mathbf{J}^N(\mathbf{a}, \mathbf{b}) = \mathcal{M}(\mathbf{a}_N, t_f) + \frac{t_f - t_0}{2} \sum_{k=0}^{N} \mathbf{L}(\mathbf{a}_k, \mathbf{b}_k) w_k \tag{4.40}$$

subject to

$$\begin{aligned}
\mathbf{A}_k(\mathbf{a}, \mathbf{b}) &= \left(\frac{t_f - t_0}{2}\right)\mathcal{F}(\mathbf{a}_k, \mathbf{b}_k) - \mathbf{c}_k = 0, \quad k = 0, 1, \ldots, N, && (4.41)\\
\mathbf{B}_k(\mathbf{b}) &= \mathbf{g}(\mathbf{b}_k) \le 0, \quad k = 0, 1, \ldots, N, && (4.42)\\
\boldsymbol{\Psi}_0(\mathbf{a}_0, t_0) &= 0, && (4.43)\\
\boldsymbol{\Psi}_f(\mathbf{a}_N, t_f) &= 0. && (4.44)
\end{aligned}$$

The LGLP method relies upon a simple conversion that maintains much of the structure of the original problem. By collocating at the LGL points the functions are evaluated without any dependence on neighboring points. Once the conversion to a parameter optimization problem is complete, the system may be solved by using an NLP solver.

V. THE LGLP GRAPHICAL USER INTERFACE

The implementation of the LGLP algorithm in MATLAB, using the optimization toolbox, requires a function file containing the state equations, constraints and cost function as input. Each state equation must be transformed and recorded in the input file. This requires a detailed knowledge of the algorithm to compute the transformation to $[-1, 1]$, a familiarity with MATLAB function construction, file input/output (I/O), and `constr.m` from the optimization toolbox. Although a MATLAB knowledge base may be assumed, and the published algorithm may be learned, we desire to provide an interface that allows a user to input information about an optimal control problem in the standard form and have the program transform and solve the problem.

A. MATLAB GUIS

GUI design has almost become synonymous with windows programming. With programming for Windows, UNIX or Apple, the operating system provides the user with a familiar point and click interface to most system functions. Most operating systems (OS) provide access to a command line interface, but as each succeeding generation of computers increasingly relies upon GUIs, portability to multiple platforms and accessibility to wide audiences will only be achieved through GUIs.

MATLAB at its core provides a command line interface to allow a C-like code to be easily written and quickly interpreted to solve mathematical problems, especially in the fields requiring numerical linear algebra. The creation of a GUI within MATLAB allows access to the strength of the command line interface while providing a dynamic interface to the solution of a problem.

MATLAB GUI design, and GUI design in general, center on several principles presented in the MATLAB Graphics and GUIs manual [Ref. 15]. A GUI should have the characteristics shown in Table V.

1. Simple

2. Consistent

3. Familiar

4. Dynamic

Table V. GUI Design Characteristics

The GUI must be simple to use, as the goal is to create an interface within MATLAB which makes solving a problem easier than using the command line interface. The project must be consistent to prevent confusion as the problem is solved. A familiar interface used throughout the GUI will accelerate the learning curve, allowing a user to quickly accomplish new tasks by mimicking completed ones. Lastly, the interface must be dynamic, to allow the user to efficiently maneuver through the solution process, changing incorrect values, and providing feedback for actions while solving the problem. A GUI must also allow the user to directly access the input files. Direct file access is required for advanced users who might not want to wade through several layers of point and click interface to change a single variable or parameter before rerunning an optimization.

The hierarchy of control for MATLAB graphics provides the starting point for manipulation of GUIs, Figure 1. Each object, figure, user interface control (uicontrol), axes, and user interface menu (uimenu) have properties and callbacks that control its appearance and function. The *GUIDE*, Figure 2, simplifies access to the handle graphics objects and allows creation of a GUI by clicking and dragging the desired components off of a visual menu. The *GUIDE* control panel provides the building

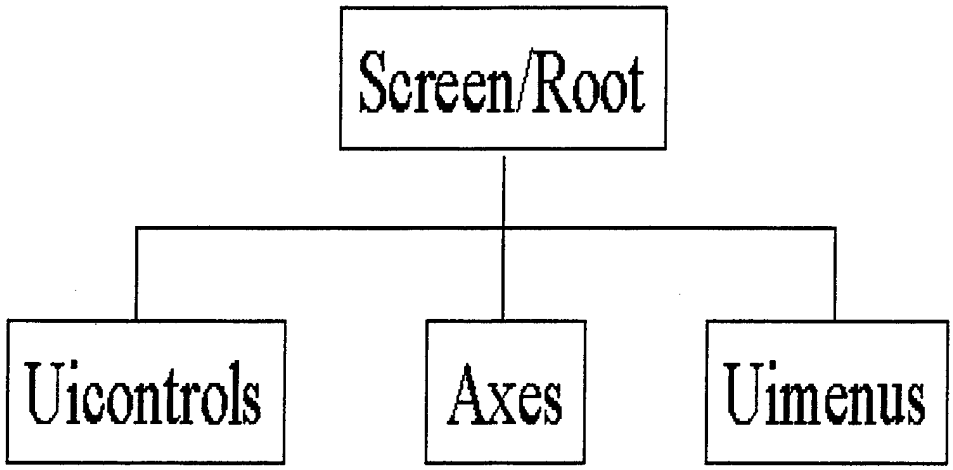

Figure 1. Hierarchy of MATLAB Graphics

blocks of the GUI and allows control of the current figures. Figures are the objects that contain all other MATLAB graphics objects.

Each of the objects will have a unique graphics handle, represented by an integer or a float. Before an object is manipulated, it must be grabbed, much like a cup must be grabbed before its contents may be drunk. Several reserved words in MATLAB make manipulating the handle graphics very simple. The abbreviations *gcf, gca*, and *gco* are used to refer to the active figure, axes and object respectively. The object that initiates a callback is referred to as *gcbo*. In windows programming, handles are used to control objects. To agree with convention, graphics handles are referred to as handle graphics. Handle graphics are covered in detail in the MATLAB documentation and In *Graphics and GUIs with MATLAB* [Ref. 13].

The alignment tool allows easy alignment of the controls of the GUI. The menu editor provides access to the figure's user defined menus, and the callback editor controls the functions that will be executed by each of the figures and the GUI controls. The property editor provides access to every property of the figure, its uicontrols and its axes. The property editor accesses each of the properties of the

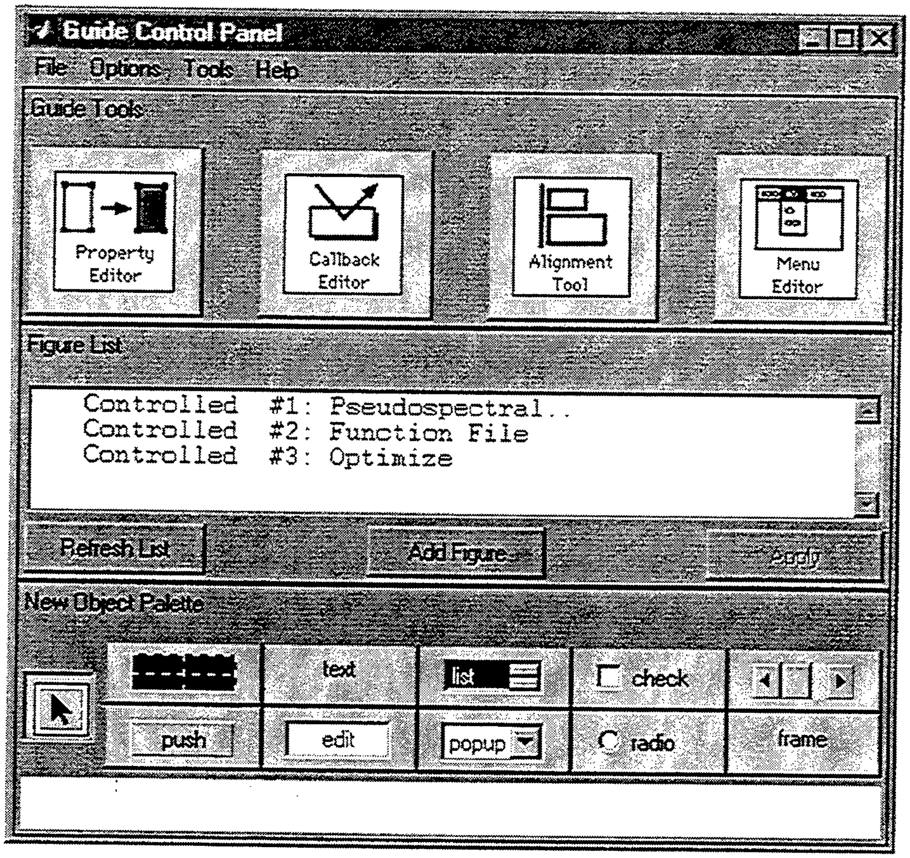

Figure 2. MATLAB Guide

objects. Their size, color, and functionality can all be edited from within the property editor.

The callback editor is the tool that creates the true functionality for the GUI. The callbacks associated with each object are shown in Table VI.

The create function works efficiently to provide default values for variables, editable fields, and other uicontrols. Each time a figure or handle graphic is created, the *create* callback function is executed. A simple *create* function call from a text edit box of `set(gcbo,'String',Names_States)` allows a variable, `Names_States`, to be written into the text edit box that had just been created and called the *create* callback.

1. Callback Function
2. Buttondown Function
3. Create Function
4. Delete Function

Table VI. GUI Callbacks

This usage of the *create* function places the current states names in the window so the user can provide initial guesses for the states and control variables. This enhances the dynamic nature of the interface by incorporating current values and not requiring redundant input of values while still clearly illustrating the current values.

The *callback* function executes when a button is pushed, a slider slid, a box checked or field edited. A simple *callback* is illustrated by the *State Equations* button on the *Function File* GUI, Figure 5. The code `StatesEquationsFile` is executed and the *State Equations* figure is displayed. Another *call back*, `close(gcbo)`, closes the *State Equations* box and returns to the *Function File* GUI.

The *delete* function may be used to clear the workspace when an object is closed. This can keep the workspace from becoming cluttered with variables that are no longer needed. The *buttondown* function activates code when the cursor is located over the edge of any object. This differs from the *callback* function, as the mouse does not click on the object, but only needs to rest over the object to activate the function call. These basic tools of the *GUIDE* allow for the rapid development of a GUI and provide a framework that can be edited by hand to further optimize or tailor the code.

Several usability features and design considerations are necessary when creating large MATLAB GUIs. *Create*functions, *callback* functions, and file operations form the backbone of the GUI. Designing the *callbacks* involves deciding to use a script, function, or in-line code for all the executable code. In-line code has the disadvantage that long sections of code are very difficult to read in the *GUIDE* environment and are not stored in the GUI m-file (ASCII), but instead in the mat-file (binary). This makes the code editable only within the *GUIDE* environment and prevents editing the code by hand. Function calls execute code the fastest, but require proper parameter passing to ensure the proper variables are passed into the function workspace. A function space is created each time a function is executed. This provides safety from overwriting variables in the main workspace, and ensure that all the necessary variables will be available when the function is executed. Writing individual functions can require a large number of files to be associated with the GUI. Scripts are slower than functions, but by using a `switch` statement in a script, numerous *callbacks* can be made through a single file to reduce the number of files included in the implementation. A `switch` statement could also be used with a function call, but the parameter passing becomes increasingly difficult as different calls are made to a function with different parameters. MATLAB does not have strict type checking, so generic parameters may be used with a function, but the code is very cryptic as the variable names will not be named according to their function. For example, a MATLAB function could take three parameters,

```
function switcher(para1, para2, type);
```

and have a `switch` statement within the function that evaluated the value of the parameter `type`. For one value of `type`, the other two parameters, `para1, para2`, could be strings. For another value, the parameters could be floats. Neither of the parameter names, `para1, para2`, provide any insight into their values.

B. GUI DESIGN

The creation of a GUI for the LGLP algorithm makes the method accessible to a wider audience of engineers and applied mathematicians. The mathematics required to perform the transformation of the problem into $[-1,1]$, the derivation and application of the differentiation matrix, and the implicit integration all provide potential stumbling blocks to easy access to the strength of the LGLP algorithm. These stumbling blocks are removed by an efficient GUI.

A GUI must have unity of purpose. The purpose of this interface is to solve optimal control problems. The interface must be simple and consistent to allow easy solution of problems and prevent the interface from distracting from the solution process. The original implementation required writing a function file outlining the states, constraints and cost functions, and then creating a function or script to set the proper variables in the main workspace and invoke the NLP solver. For simplicity we kept this two step process in the design of the GUI, Figure 3.

The first design task for the new GUI for the LGLP algorithm was to sketch the interface. The two main tasks, creating the file, and setting the variables, should be separated for clarity. Also, once the function file has been created the optimization window must allow for repeated runs in multiple sessions without recreating the file code.

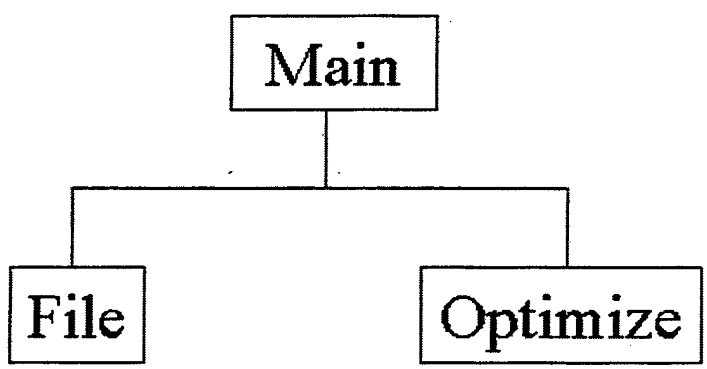

Figure 3. Initial GUI Design

Figure 4 serves as the starting point for the GUI. *Typing* opt *at the command line will execute the GUI.* In Figure 5, the *Function File* figure creates the input files

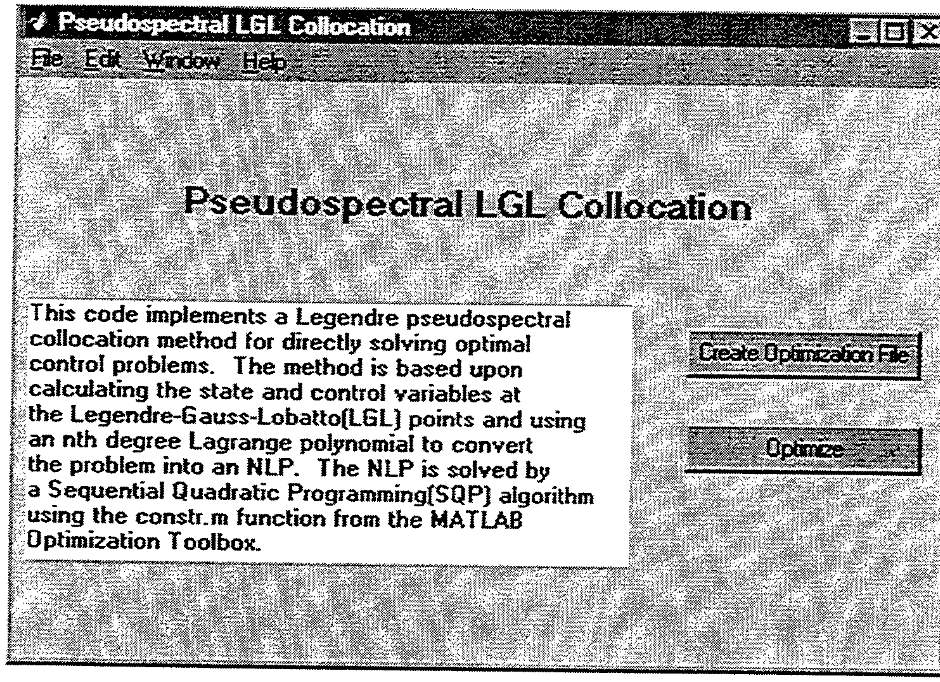

Figure 4. LGLP GUI Welcome Figure

necessary for NLP solver, `constr.m` [Ref. 16]. The GUI must be able to convert an optimal control problem, with fixed or free final time, all of its state equations, control constraints, equality and inequality constraints, and parameters into an NLP.

Figure 6 solves the optimization problem. The optimization of the input file requires the parameters for `constr.m` to be selected. Also, the interval size for the discretization must be selected, the values for any parameters must be set, and the initial guesses must be provided for each of the states, controls, and final time.

Another design decision was to ensure that default values would be created to prevent errors if not all of the possible inputs were selected. By limiting the *callbacks* executed by the main figure's children to merely receiving input, the main computation could be isolated to the main GUI figures. This ensured that if all of the variables had been input, then the GUI would run properly.

The main input for the `constr.m` optimization algorithm is a file that contains

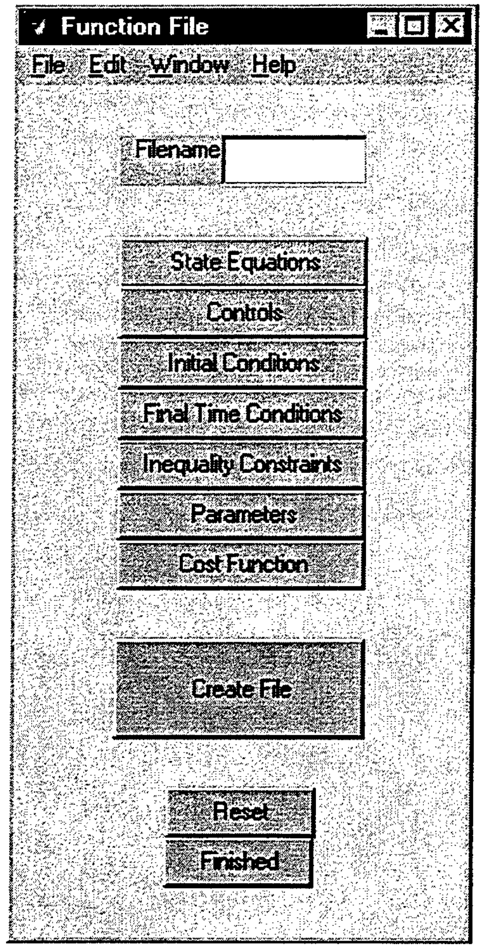

Figure 5. Function File Creator Figure

the state equations, constraints, and cost function. This requires MATLAB to create and manipulate a file through use of file I/O that mirrors ANSI C or C++. A second MATLAB file feature that simplified the GUI was the mat-file constructs which saves the values of the variables used during the optimization. Every time an m-file was created, a sister mat-file was created with the variables declarations. This allowed all of the variables associated with a function file to be loaded into the workspace with *create* functions. Once a filename has been selected, the GUI attempts to load a corresponding mat file. A flag returned by MATLAB when the mat function does not exist, which occurs the first time a function is created, prompts the initialization

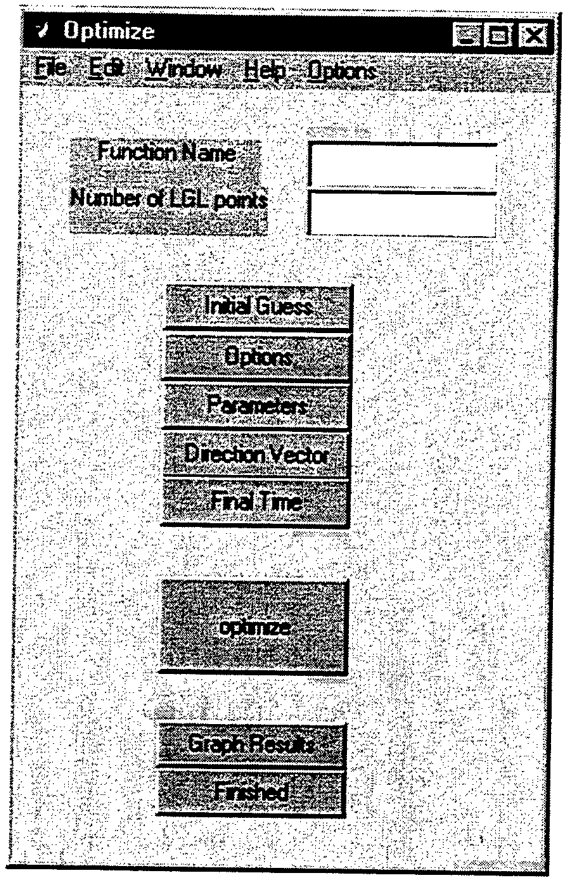

Figure 6. Optimization Figure

of all the variables required by the GUI. The diary function was also used to create an output file after the optimization without requiring redundant file I/O. The diary file captures all the optimization results that display in the MATLAB workspace, creating a record of each optimization run. By choosing the append option for file I/O, each optimization run is appended to the same diary file. The clock function is used to record the date and time of day of optimization in the diary file to distinguish between runs in the file, see Appendix A,B, and C.

Every editable textbox in a GUI takes a string as input. MATLAB does not allow a particular textbox to be declared as a float. This convention of all inputs being declared as strings motivates a style of coding that takes advantage of this structure. Utility functions designed to work with strings and accomplish simple, repetitive tasks can streamline the coding of a GUI. A multiline input box returns a matrix of string variables. In a GUI with two multiline textboxes, we assign the strings in one box as variable names and use the inputs from the other box for values of these variables. The function `declare.m` is a simple utility function that takes two matrices of strings and combines the strings into code that will assign the names of the variables to the values, see Appendix A. The `eval` command is used in MATLAB to execute the commands contained in a string variable. The `eval` is used to execute the string matrices that are returned from `declare.m`.

Two other utility functions were created with this GUI, `optcall.m` and `costcall.m`. Both accept parameters as input to create function calls to be evaluated by the GUI. The function `optcall.m` creates a function call for `constr.m`, and `costcall.m` evaluates the cost function for the files created with the *File Creator* GUI.

Creating help files for a GUI is an important part of the design. MATLAB has two features for creating help, comments within an m-file and the `Contents.m` file. The `declare.m` code in Appendix A illustrates the MATLAB comment help convention. The initial comments in an m-file after the function header are included by MATLAB as help for the function. When `help declare` is typed at the command line the initial commented lines of code in the file is displayed. MATLAB displays the comments until the first line after the function header that does not include the comment character.

The second convention is the `Contents.m` file. The comments in the `Contents.m` file, are displayed when the `help` command is called with a directory as its input argument. When a directory does not include a `Contents.m` file, the first help line (H1) of each m-file in the directory is displayed. Creating a `Contents.m` file provides another method for the user to get help on a GUI, see Appendix B. All GUI files may have comments and help lines built into their code. However, when a GUI is edited with the *GUIDE*, any comments are erased and a standard comment explaining that this code is a computer generated object is inserted. This severely limits the usefulness of adding any comments lines within the GUI. By placing the comments that would be included within the GUI in the `Contents.m` file, the comments will be preserved regardless of the number of times a GUI is modified.

A simple example problem will illustrate the use of the GUI. The GUI was laid out to encourage all text boxes to have values entered before the various buttons were pushed. All the buttons are designed to be executed from top to bottom, with a familiar *Finished* button in the bottom of each figure.

VI. A GUI TUTORIAL-THE CART PROBLEM

In a recent paper by Conway and Larson, "Collocation Versus Differential Inclusion in Direct Optimization", the following problem was posed [Ref. 8]. A cart is placed on a track initially at rest. An external force $u(t)$ is applied to the cart of unit mass. The cart is subject to drag depending linearly on the velocity. The system equations are

$$\dot{x}_1 = x_2 \tag{6.1}$$

$$\dot{x}_2 = -x_2 + u. \tag{6.2}$$

The force $u(t)$ is to be applied so as to satisfy a terminal constraint at the final time t_f which is a combination of position and velocity

$$\psi_f = ax_1(t_f) + bx_2(t_f) - c = 0$$

while minimizing the integral of the square of the control

$$J = \int_{t_0}^{t_f} u^2 dt. \tag{6.3}$$

With the values of the constants chosen as $a = 1.0$, $b = -2.694528$, and $c = -1.155356$, and $t_f = 2$, the value of the objective function is $J = 0.577678$ [Ref. 8]. This problem provides an easy tutorial as the analytic solution is known and the problem is quickly optimized by `constr.m`.

A. FILE CREATION

Typing `opt` at the command line starts the GUI and allows access to *Create Optimization File* button. Pressing this button starts the input file creation process. The implementation begins with the creation of a file that contains the constraints, state equations, and cost function for the problem. The first step is always to enter a filename for the file; see Figure 7. Entering the filename will set the current filename and will also check to see if a MATLAB data file exists for this filename. If a file exists, the file will be loaded. This loads all the variables associated with the problem into the MATLAB workspace.

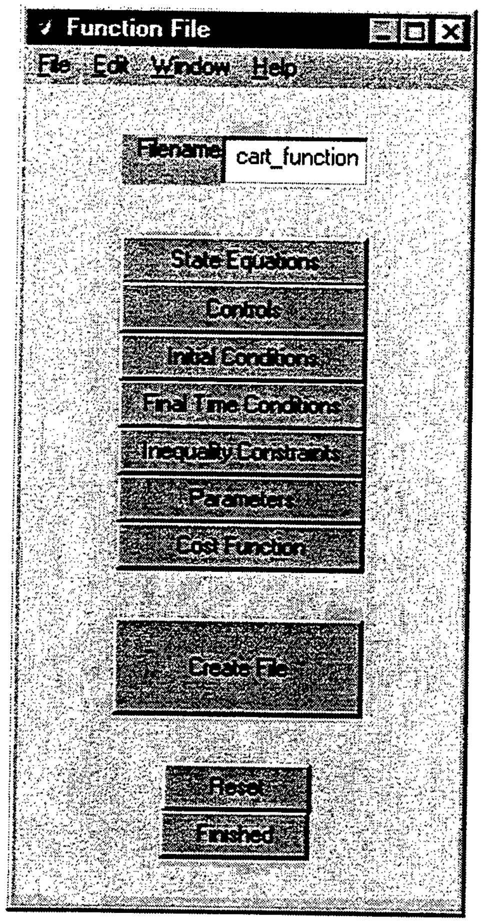

Figure 7. Cart Problem File Creator - GUI

After the filename for the input file to `constr.m` has been chosen, buttons execute additional figures to collect the necessary variables and parameters for the problem. Next the state variables, x_1 and x_2, are entered into the GUI implementation of our method. Pressing the *State Equations* button will open a window for inputting the state equations for x_1 and x_2, see Figure 8. The state equations are entered with

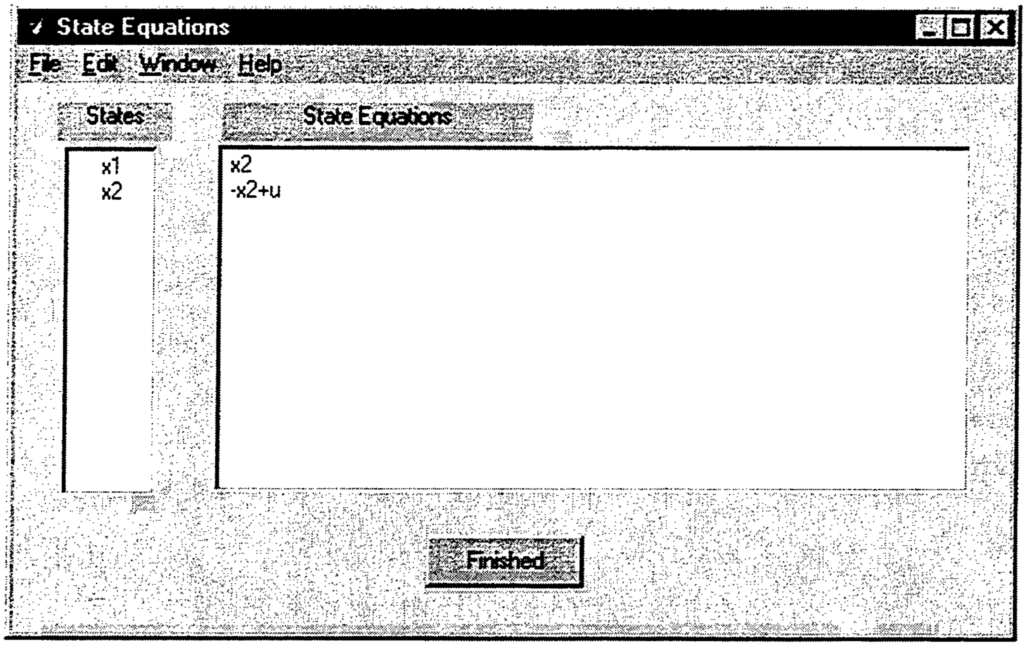

Figure 8. Cart Problem State Equations - GUI

the $\dot{\mathbf{x}}$ assumed as the left hand side of the equality. Here we must make a transition into MATLAB numerical notation. All entries must be legal MATLAB expressions. Of special note is the difference between matrix multiplication and element-by-element array multiplication or dot multiplication. Dot multiplication requires the use of the $.*$ construct to ensure dimensional agreement within MATLAB. The states, x_1 and x_2, are both $n \times 1$ vectors. Each component corresponds to a discrete time step between the initial and final times. A simple system of equations arises from choosing to use three LGL points for this example. The state equations, (6.1 -6.2), can be expressed as a systems of six equations of scalar variables:

$$\dot{x}_1(1) = x_2(1) \tag{6.4}$$

$$\dot{x}_1(2) = x_2(2) \tag{6.5}$$

$$\dot{x}_1(3) = x_2(3) \tag{6.6}$$

$$\dot{x}_2(1) = -x_2(1) + u(1) \tag{6.7}$$

$$\dot{x}_2(2) = -x_2(2) + u(2) \tag{6.8}$$

$$\dot{x}_2(3) = -x_2(3) + u(3). \tag{6.9}$$

This form of the systems of equations illustrates that the operations are conducted component-wise. A slightly different equation illustrates dot multiplication. Let the second state equation be

$$\dot{x}_2 = -x_2 x_1 + u. \tag{6.10}$$

The second state equation can be expressed as

$$\dot{x}_2(1) = -x_2(1)x_1(1) + u(1) \tag{6.11}$$

$$\dot{x}_2(2) = -x_2(2)x_1(2) + u(2) \tag{6.12}$$

$$\dot{x}_2(3) = -x_2(3)x_1(3) + u(3). \tag{6.13}$$

The operations are component-wise in this system of equations. To properly represent this system in MATLAB we use dot multiplication, `.*`. The proper MATLAB input for the GUI would be `-x2(n).*x1(n)u(n)`. MATLAB's dot multiplication performs the necessary component-wise operations on the state and control vectors without requiring that the continuous equations be converted into a system of discrete equations.

When using the Windows OS, a CTRL ENTER must be used to record a multi line input. MATLAB only distinguishes between single and multi-line inputs and the CTRL ENTER tells MATLAB the user has finished with the text edit box. Extra lines should not be added between the values or after the values. The UNIX OS does not require a CTRL ENTER, but if a standard ENTER is used after the last entry, and extra empty entry will be added to the input vector.

After closing the state equations window, the *Controls* variable button is pressed and the control u is entered in the *Control Variables* window, see Figure 9. To set the variables x_1 and x_2 to the initial value of $x_1 = 0$ and $x_2 = 0$, the first

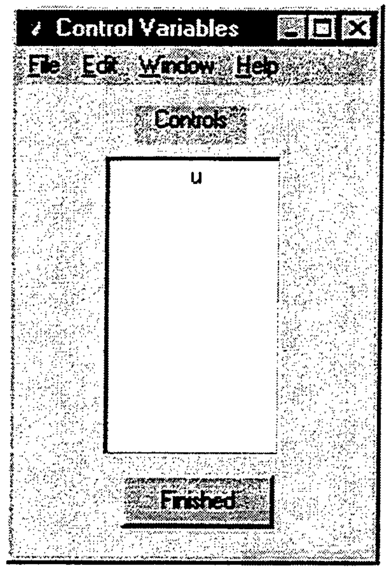

Figure 9. Cart Problem Controls - GUI

discrete value of the x_1 and x_2 vectors must be set to zero. This is accomplished by writing the equation in a standard NLP form with the equation set equal to zero.

$$x_1 = 0 \quad \equiv \quad \texttt{x1(1)} \tag{6.14}$$

$$x_2 = 0 \quad \equiv \quad \texttt{x2(1)}. \tag{6.15}$$

For a slightly different initial condition, $x_1 = 1$, we enter

$$\texttt{x1(1)} \quad - \quad \texttt{1}. \tag{6.16}$$

It is assumed (6.16) is set equal to zero. This format is standard for NLP formulations and will be used for all the constraints entered into the GUI. The two initial conditions for the cart problem, $x_1 = 0$ and $x_2 = 0$, are entered in the *Initial Conditions* window in Figure 10.

Next, the final time boundary conditions are entered along with a selection of fixed or free final time, see Figure 11. The discretized final time variables are represented as the last component of the state vectors, in this example `x1(n)` and `x2(n)`. Within the GUI final time constraints and initial time constraints are both

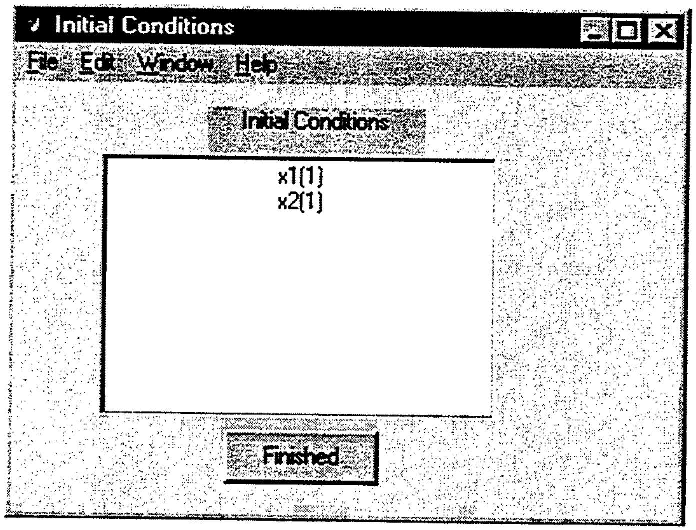

Figure 10. Cart Problem Initial Conditions - GUI

treated as scalar constraints. The boundary conditions are input in two figures for clarity only. The difference between boundary conditions is transparent to the NLP.

Next, the three parameters a, b, and c are entered in the parameters window, see Figure 12. Values needed by the function while is it being optimized must be passed through the parameters window. The values of the parameters are free at the time of the file creation. The values are set in the *Optimization* figure, allowing for easy manipulation of the constants while solving the problem.

Finally, the cost function is entered. The type of cost function is selected, in this case Lagrange. The cost function requires raising the control to the second power, see Figure 13. This is accomplished by component wise multiplication or exponentiation. As explained earlier, the equation requires the `.^` construct to square the control. These operations are done element-by-element and are expressed in MATLAB as `u.^2` or `u.*u`.

The selection of a type of cost function controls the number of inputs allowed to the *Cost Function*. If either the Mayer or Lagrange type are entered, only one input is allowed. When the Bolza type cost function is selected, two inputs are

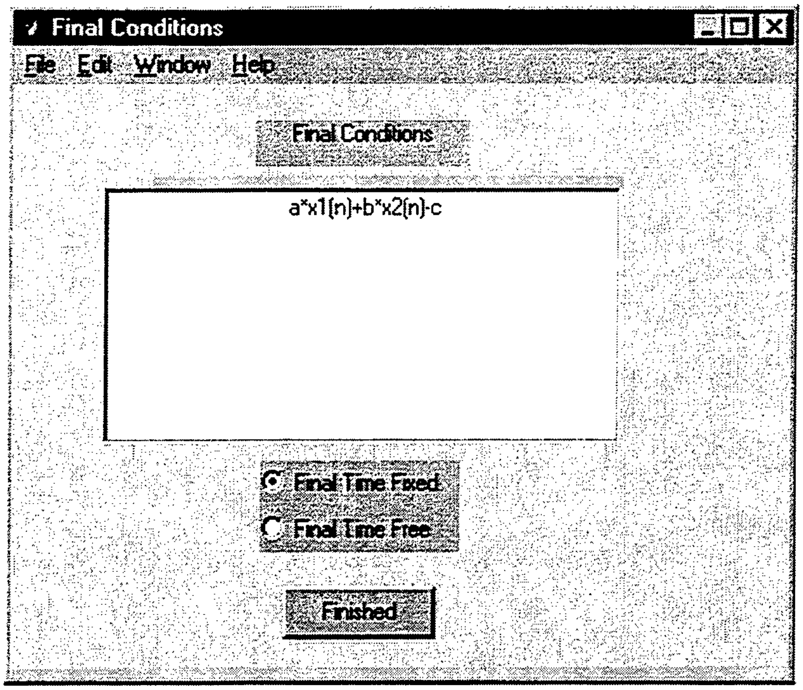

Figure 11. Cart Problem Final Conditions - GUI

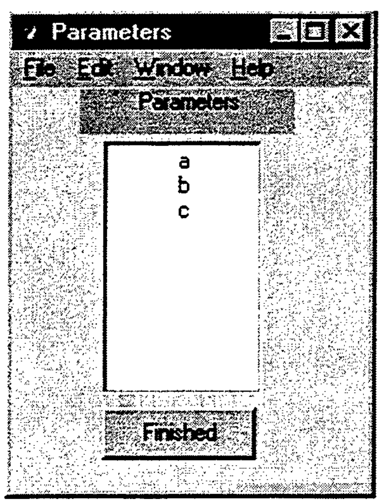

Figure 12. Cart Problem Parameters - GUI

expected. With a Bolza cost function, the first line input is the integral portion of the cost function, and the second line of the input is the scalar portion. The proper operation, addition or subtraction, must be included before the scalar portion of a Bolza cost function. The proper input of the Bolza cost function $\int_{t_o}^{t_f} u^2\,dt + 1$ would be

```
u.^2
+1.
```

For the Mayer cost function t_f the equivalent Lagrange cost function would be `ones(1,n)` to allow for the numeric integration from 0 to t_f. The MATLAB code `ones(1,n)` creates an n dimensional vector with each component equal to one. *The integrand in the integral portion of the cost function must always be a vector quantity, both for the Lagrange and the Bolza cost functions.*

After all values have been entered the file is ready to be created. When the

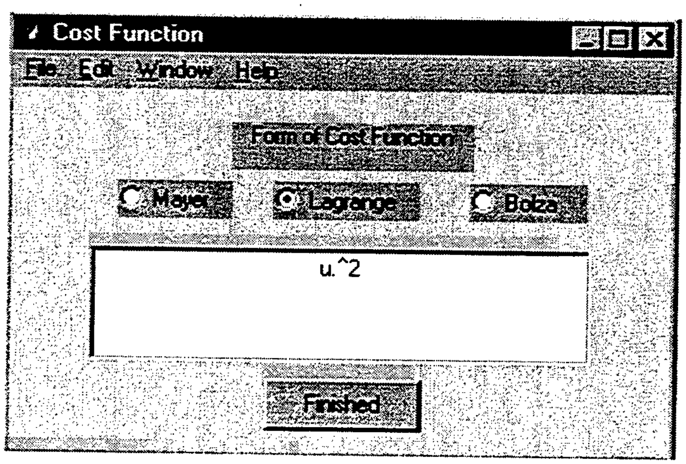

Figure 13. Cart Problem Cost Function - GUI

Create File button is pushed the input file is created and a status window indicates the task completion, see Figure 14.

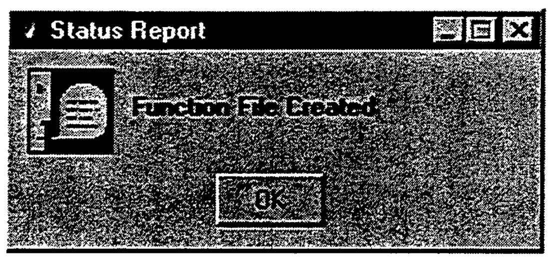

Figure 14. Cart Problem File Creation Status - GUI

B. OPTIMIZATION

Once the file has been created, the optimization window is used to enter the optimization parameters. The name of the file to be optimized is entered along with the number of LGL points to be used in the optimization, see Figure 15. The larger the number of LGL points the finer the discretization. Once again, fill in the blanks before pushing any buttons. The intuitive flow for the GUI is from top to bottom, and the first two tasks completed need to be the selection of the *File Function Name* and the *Number of LGL points.* The number of LGL points may be changed at any time. However, changing the name of the function will load the MATLAB data file associated with the new function and overwrite the current variable values.

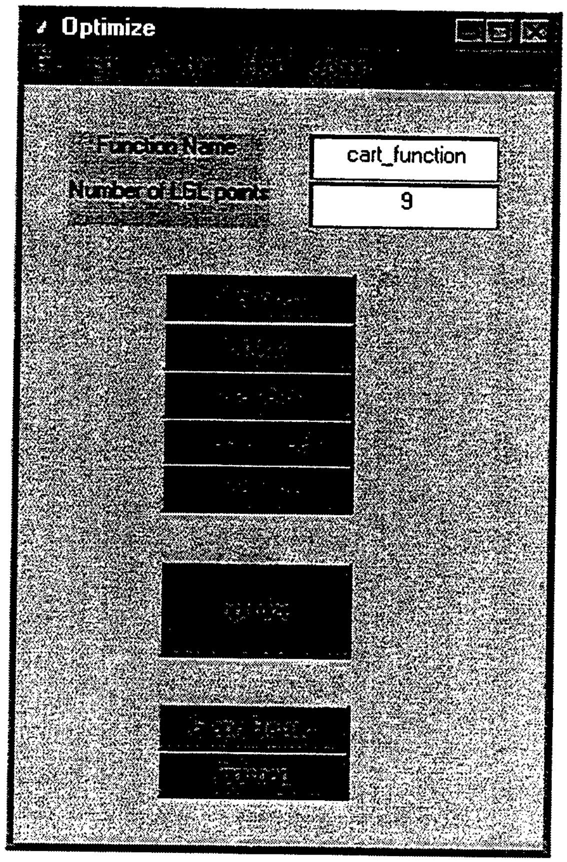

Figure 15. Cart Problem Optimization - GUI

After pressing the *Initial Guess* button the initial guess for the states and the controls can be entered. The cart problem is very robust, so we use a random number generating function to provide a random initial guess. A value for final time, t_f, is entered along with the initial guesses. If t_f is free, this is an initial guess. If t_f is fixed, the value for t_f is entered. In this case, the fixed final time value is $t_f = 2$, see Figure 16.

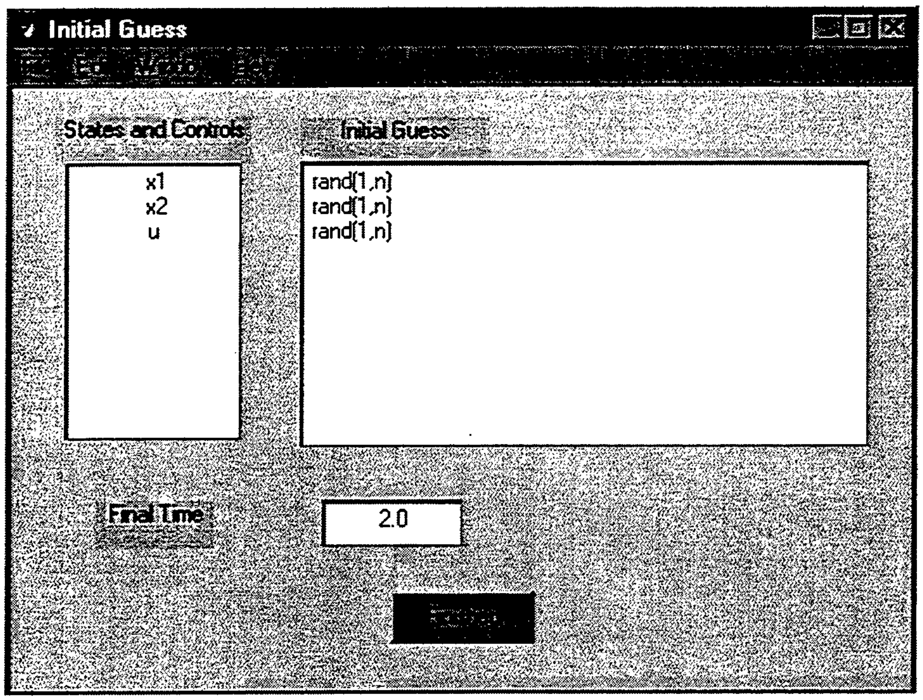

Figure 16. Cart Problem Initial Guess - GUI

The options for the `constr.m` function are changed by the *Options* button. This allows us to set upper and lower bounds, number of iterations, and number of output parameters. We can also check the number of equality constraints generated for the problem, see Figure 17. There are four groups of output parameters. The listbox allows the selection of the desired set. The most common output selection is only *Xopt* because the addition of output parameters slows the optimization. *Xopt* is the vector of optimization variables: the states, controls, and t_f for a free final time problem. The value of the MATLAB output parameter vector, the problem's Hessian, and lambdas are the additional output parameters. If the lambdas were the Lagrange multipliers for the NLP, they would be very useful for costate estimation. Unfortunately the lambdas returned by `constr.m` after the optimization are not the Lagrange multipliers necessary for costate estimation. The display intermediate results check box toggles the display of the intermediate steps during the optimization process. The upper and lower bound can be left blank and are then passed to `constr.m` as empty.

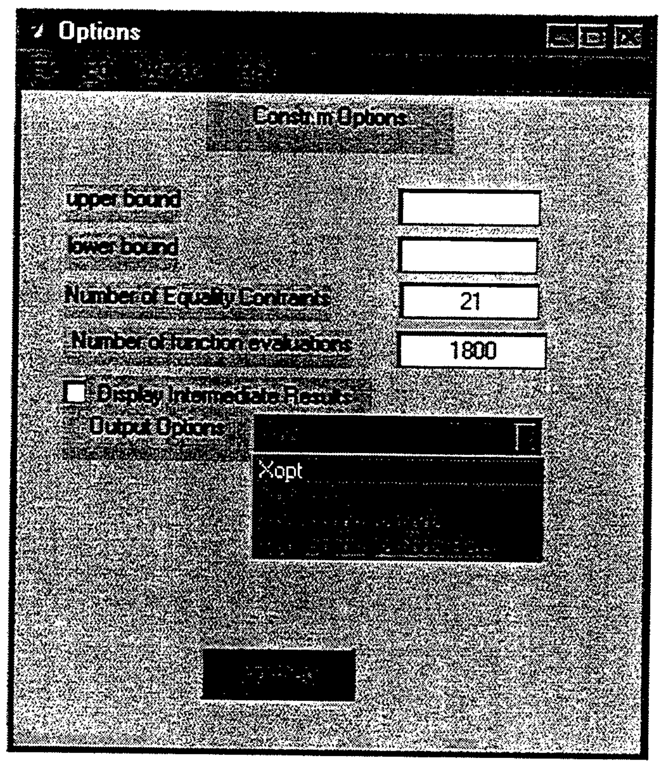

Figure 17. Cart Problem Options - GUI

Lastly, we input the values for the parameters designated during the file creation, see Figure 18. The *Direction Vector* option allows designation of the initial search vector direction. The default is set to empty. The *Final Time* figure shows a set of radio buttons to check the status of the final time, Figure 19. If this button

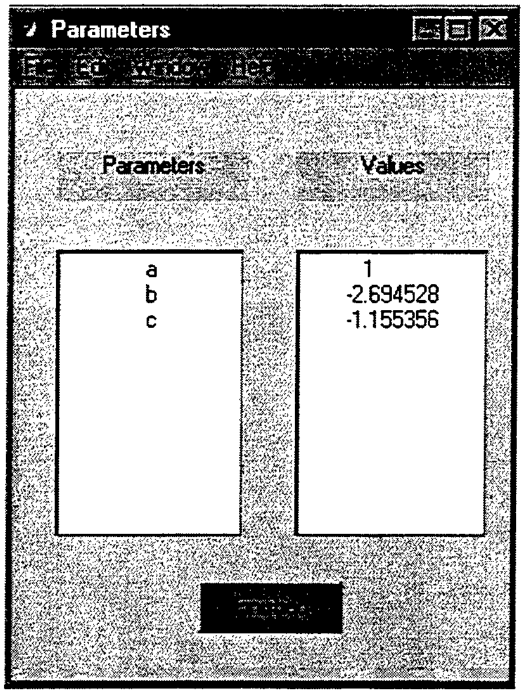

Figure 18. Cart Problem Parameter Values - GUI

Figure 19. Cart Problem Final Time - GUI

does not agree with the desired value, the function file should be recreated to ensure the proper restriction on final time.

Once the *optimize* button has been pushed, the values of the states and controls will be output to the MATLAB command window, along with the time elapsed to run the optimization, and the final value of the cost function. In addition, an output file named to agree with the input file is created. In this example the output file is `cart_function.out`. Four files are created during the optimization: `cart_function.m`, `cart_functionopt.m`, `cart_function.mat` and `cart_function.out`. `Cart_function.m` is the m-file that contains the constraints, state equations and the cost function. The `cart_functionopt.m` is a script that will declare variables and call the NLP solver. The `cart_function.mat` is the binary file that saves the values of the variables used in the optimization. The `cart_function.out` file is the output file that captures the diary of the output of the optimization to the MATLAB workspace. The GUI implementation's results agree with the published results [Ref. 8]. The `cart_function.m` and `cart_function.out` are included in Appendix A.

VII. THE ORBIT TRANSFER PROBLEM

This maximum radius orbit transfer problem is taken from the text by Bryson and Ho [Ref. 14]. Given a constant-thrust rocket engine, T = thrust, operating for a given length of time, t_f, find the thrust direction history, $\Phi(t)$, to transfer a rocket vehicle from a given initial circular orbit to the largest possible circular orbit. The variables are defined as

$$r = \text{radial distance of spacecraft from attracting center}$$

$$u = \text{radial component of velocity}$$

$$v = \text{transverse component of velocity}$$

$$m_0 = \text{mass of spacecraft}$$

$$\dot{m} = \text{fuel consumption rate (constant)}$$

$$\Phi = \text{thrust direction angle}$$

$$\mu = \text{gravitational constant of attracting center}$$

The problem can be stated as finding $\Phi(t)$ to maximize $r(t_f)$ subject to the state equations

$$\dot{r} = u \tag{7.1}$$

$$\dot{u} = \frac{v^2}{r} - \frac{\mu}{r^2} + \frac{T sin\Phi}{m_0 - |\dot{m}|t} \tag{7.2}$$

$$\dot{v} = -\frac{uv}{r} + \frac{T cos\Phi}{m_0 - |\dot{m}|t} \tag{7.3}$$

with initial conditions

$$r(0) = r_0$$

$$u(0) = 0$$

$$v(0) = \sqrt{\frac{\mu}{r_0}}$$

and final time conditions

$$u(t_f) = 0$$

$$v(t_f) - \sqrt{\frac{\mu}{r(t_f)}} = 0.$$

This problem can be solved with fixed final time as a maximum orbit distance problem, or with free final time to find the minimum time to a specified final orbit. Solving the problem as a maximum radius orbit transfer problem with fixed final time, the cost function in Mayer form is

$$J = -r(t_f). \tag{7.4}$$

Equation 7.4 represents maximizing the radius by minimizing the negative of the radius. Before finding the optimal solution, we must set values for the parameters or constants in the problem. Let

$$\begin{aligned} m_0 &= 1 \\ \dot{m} &= -0.0749 \\ T &= 0.1405 \\ \mu &= 1 \\ r_0 &= 1. \end{aligned}$$

A. FILE CREATION

The implementation for this problem mirrors the cart problem of Chapter VI. The state equations are more complex than those required for the cart problem.

```
r = u
u = (v.^2)./r -(mu)./(r.^2) +T*sin(phi)./(m0-m1*t)
v = (-u.*v./r + T*cos(phi)./(m0-m1*t)
```

are entered as the state equations into the *State Equations* GUI, see Figure 20. All the variables in the state equations will be defined in the creation of the GUI, with one exception. The variable t is predefined within the GUI to represent time. The

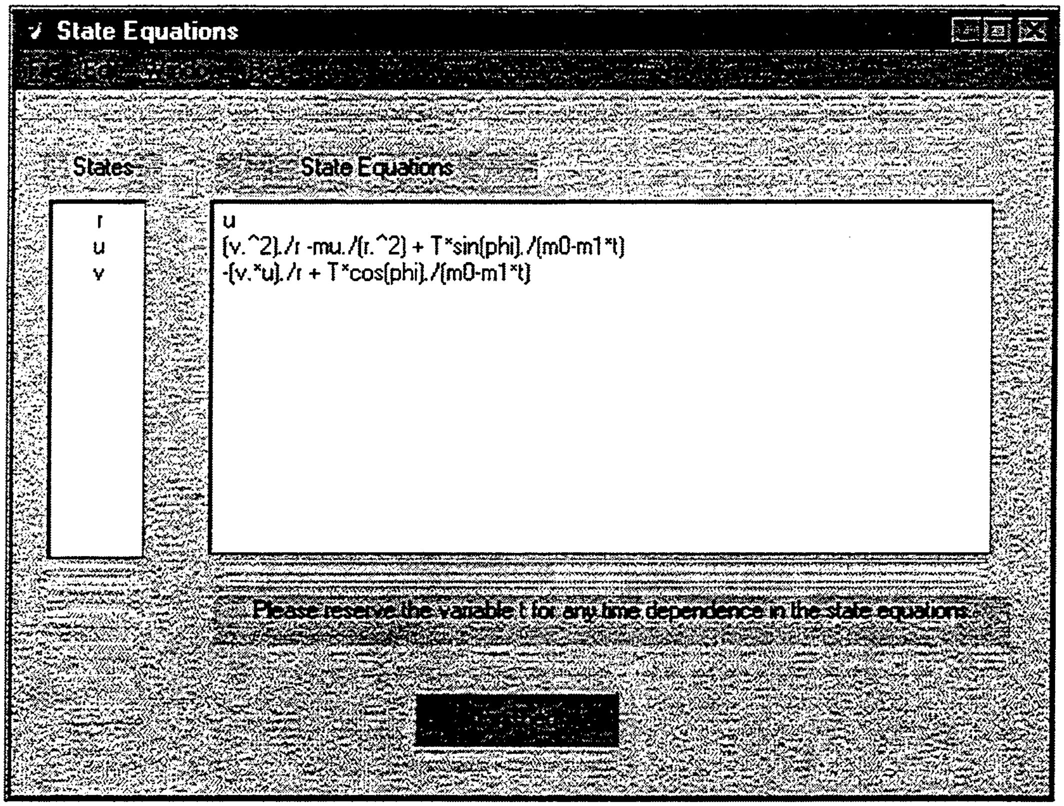

Figure 20. Maximum Radius State Equations - GUI

values of t are the LGL points scaled to the interval $[t_0, t_f]$. After entering the controls and the initial conditions, see Figures 21 and 22, the final time conditions are entered. Figure 23 shows the final time boundary conditions when final time is fixed. This problem also has inequality constraints which must be entered. The inequality constraints are restrictions on the allowed valued for the control, ϕ. The values for ϕ are limited to $-\pi \leq \phi \leq \pi$. Since `constr.m` only allow inequalities of the form $g(n) \leq 0$, the expression must be transformed into the normal form before it may be entered into the GUI.

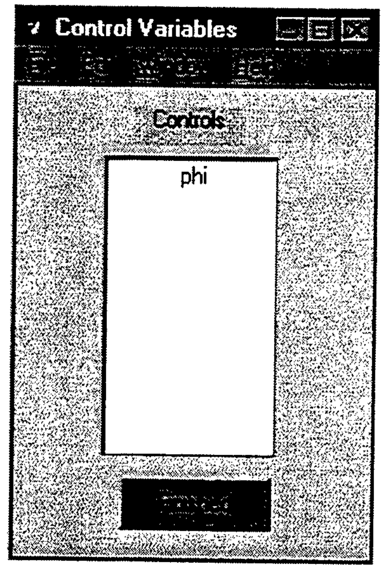

Figure 21. Maximum Radius Controls - GUI

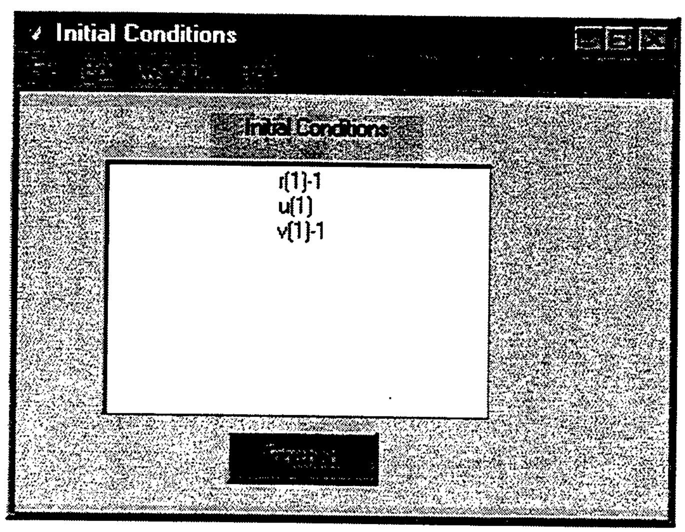

Figure 22. Maximum Radius Initial Conditions - GUI

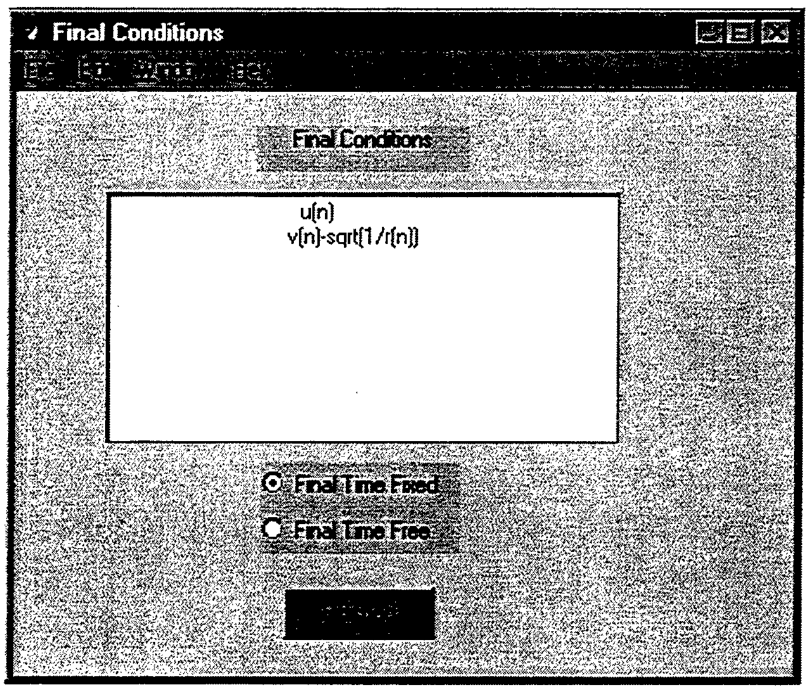

Figure 23. Maximum Radius Final Conditions - GUI

$$-pi \leq \phi \;\leftrightarrow\; \phi - pi \leq 0 \qquad \text{and} \tag{7.5}$$

$$pi \geq \phi \;\leftrightarrow\; \phi - pi \leq 0 \tag{7.6}$$

see Figure 24. The constants are entered in the parameters GUI, Figure 25. The

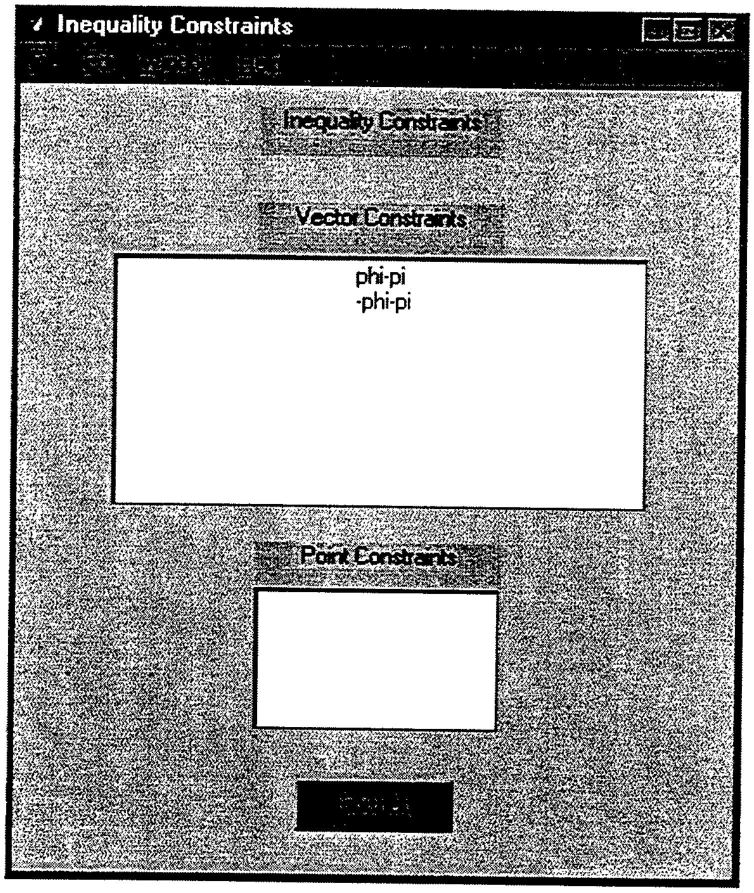

Figure 24. Maximum Radius Inequality Constraints - GUI

final step is the creation of the cost function as shown in Figure 26. The discrete cost function for the Mayer form of the cost function is

$$J = -r(n).$$

The optimization file created and the output from the optimization for the maximum radius problem are located in Appendix B.

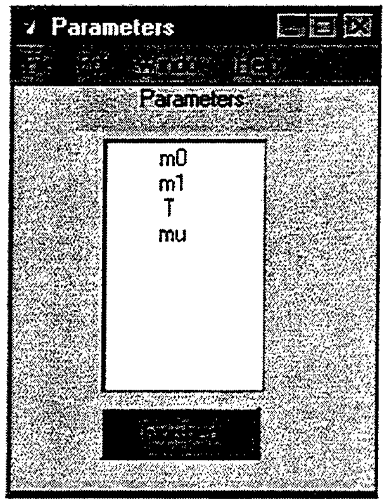

Figure 25. Maximum Radius Parameters - GUI

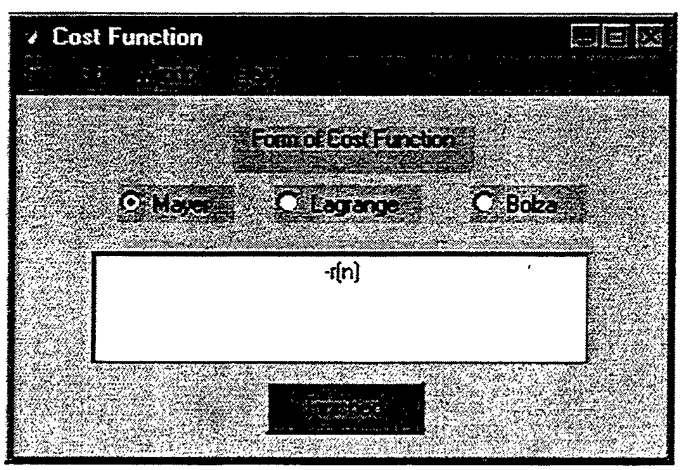

Figure 26. Maximum Radius Cost Function - GUI

B. OPTIMIZATION

Before optimizing the problem, the initial guesses must be entered, Figure 27, The initial guess for each of the states and controls may be a column or row vector,

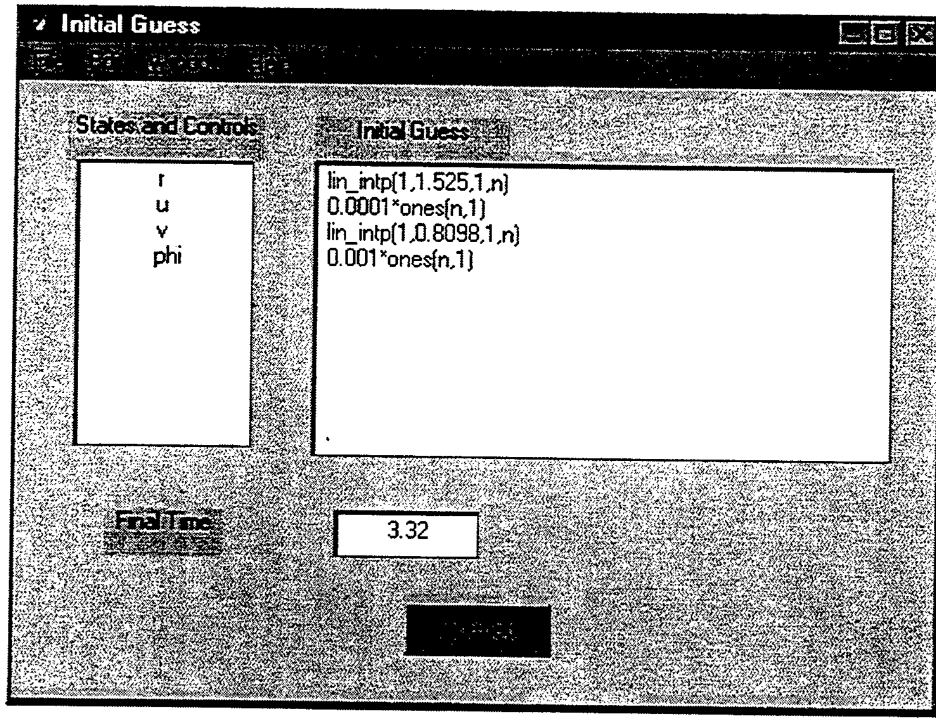

Figure 27. Maximum Radius Initial Guess - GUI

but they must all agree in dimension. The `lin_intp` function input as the initial guess for r, outputs a linear guess with values in $[1, 1.525]$ as an $n \times 1$ row vector. Once the first initial guess is input as a row vector, all others guesses must be row vectors also. The parameters are set to their constant values, see Figure 28. The *Options* button can be used to check the number of equality constraints and increase the number of iterations allowed. The *Final Time* button can be used to check for the correct time characteristics, Figure 29. The number of equality constraints in the NLP must be equal to the number of state equations and boundary conditions.

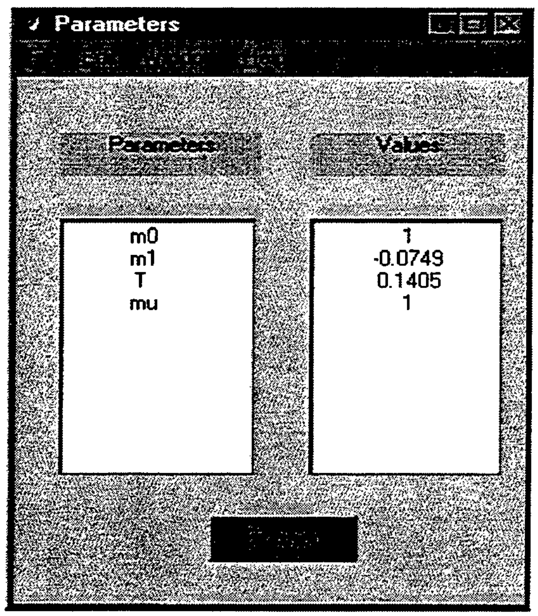

Figure 28. Maximum Radius Problem Parameters - GUI

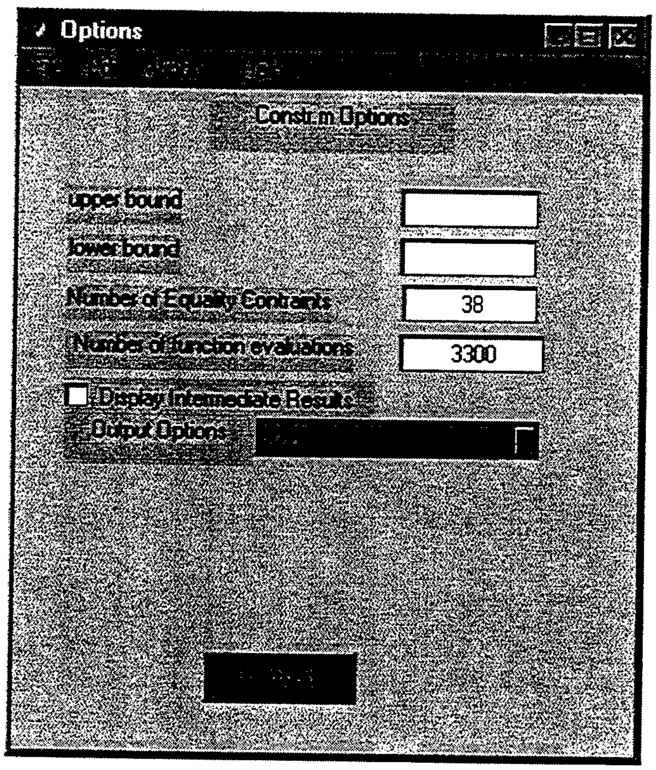

Figure 29. Maximum Radius Problem Options - GUI

Having chosen 11 LGL points, the number of equality constraints are

$3n = 33$	state equations
3	initial/boundary conditions
2	final/boundary conditions
--	
38	equality constraints

Often the default number of function evaluations will be too small for the problem to converge. The number displayed in the *Options* GUI is the default for `constr.m`. It is easily increased by adding zeros to the displayed number. The *optimize* button is pressed to begin the optimization. A diary file, `bho.out`, resulting from the optimization run with 11 LGL points is included in Appendix B.

C. CHANGING THE FILE

Changing the problem to a free final time problem, where the final radius is fixed and the objective function switches to optimizing time, is accomplished by several simple changes to the function file. By pressing the *Final Conditions* button, adding `r(n) - radius` as a final time condition, and changing the final time to free, the problem has been slightly modified to the new variant, see Figure 30. The

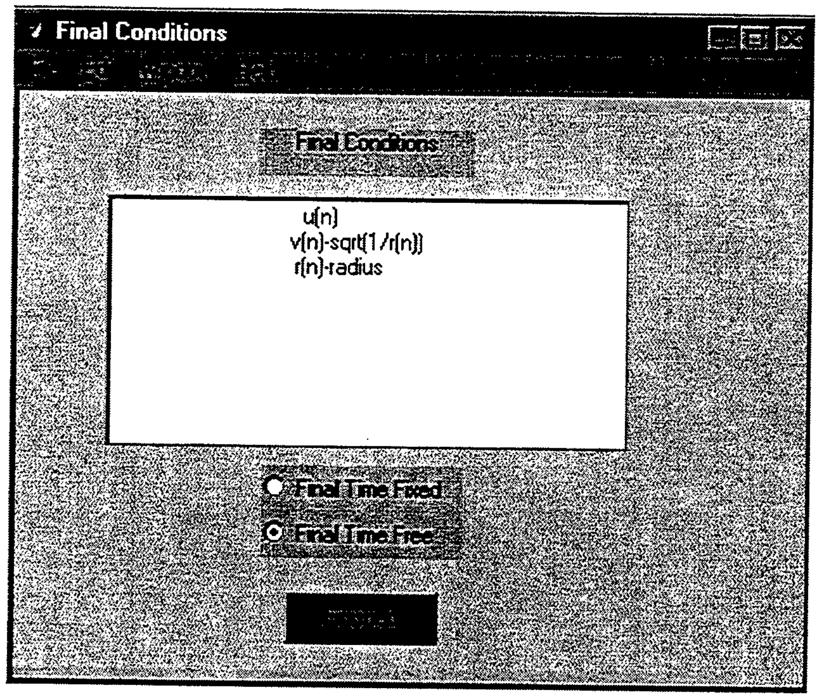

Figure 30. Optimal Time Problem Final Conditions - GUI

Inequality Constraints must be modified to include a non-negativity constraint on t_f, see Figure 31.

After adding the parameter `radius` to the parameter list, Figure 32, and changing the cost function to t_f, Figure 33, the new problem has been formulated. Pressing the *Create Function* button creates the new m-file and the problem is ready to be optimized. The m-file generated is included in Appendix C.

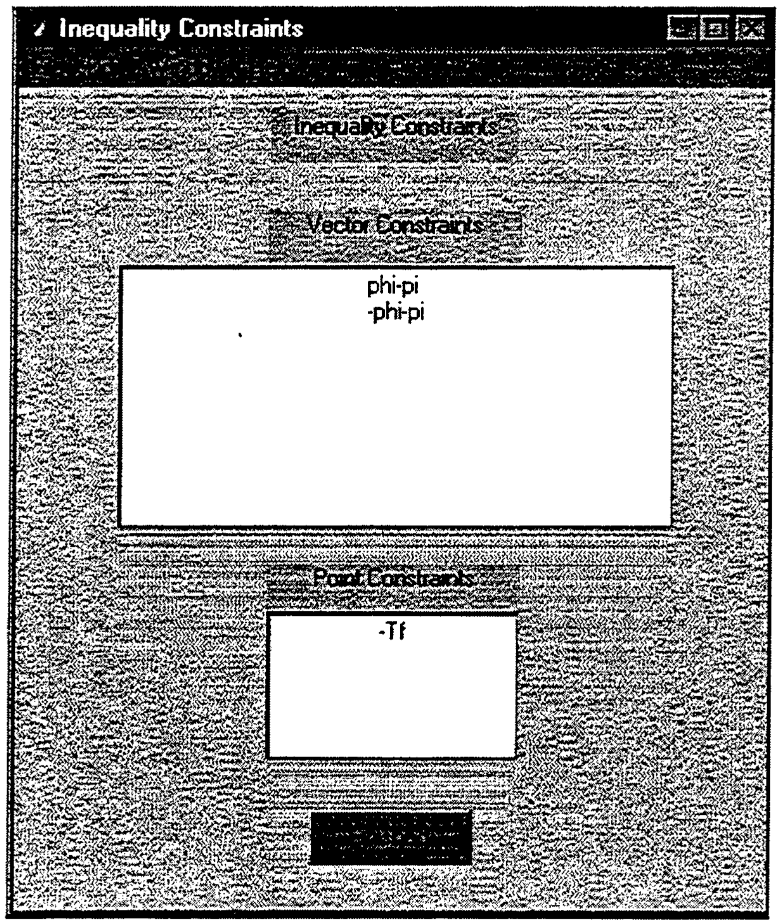

Figure 31. Optimal Time Problem Inequality Constraints - GUI

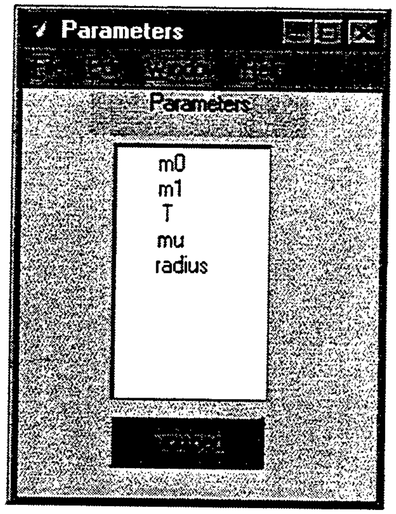

Figure 32. Optimal Time Problem Parameters - GUI

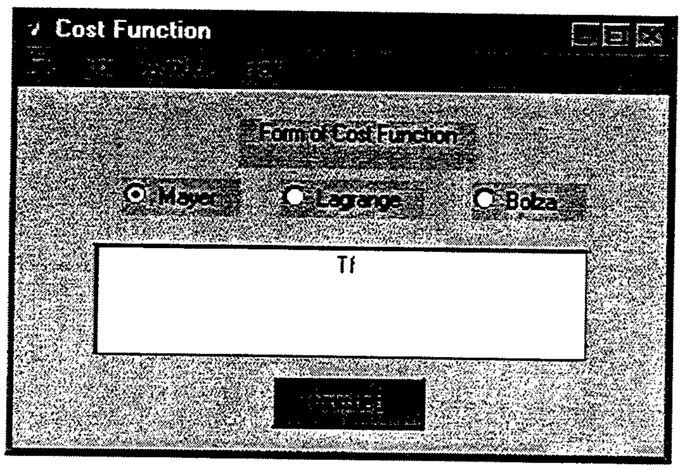

Figure 33. Optimal Time Cost Function - GUI

D. A SECOND OPTIMIZATION

Suppose we want to use the data from the last optimization as the initial guess for solving this slightly modified problem. Enter the values from the last optimization run into the *Initial Guess* figure, see Figure 34. Choose a value for the parameter radius, $radius = 1.345$, and enter the value in Figure 35. Check the *Final Time*

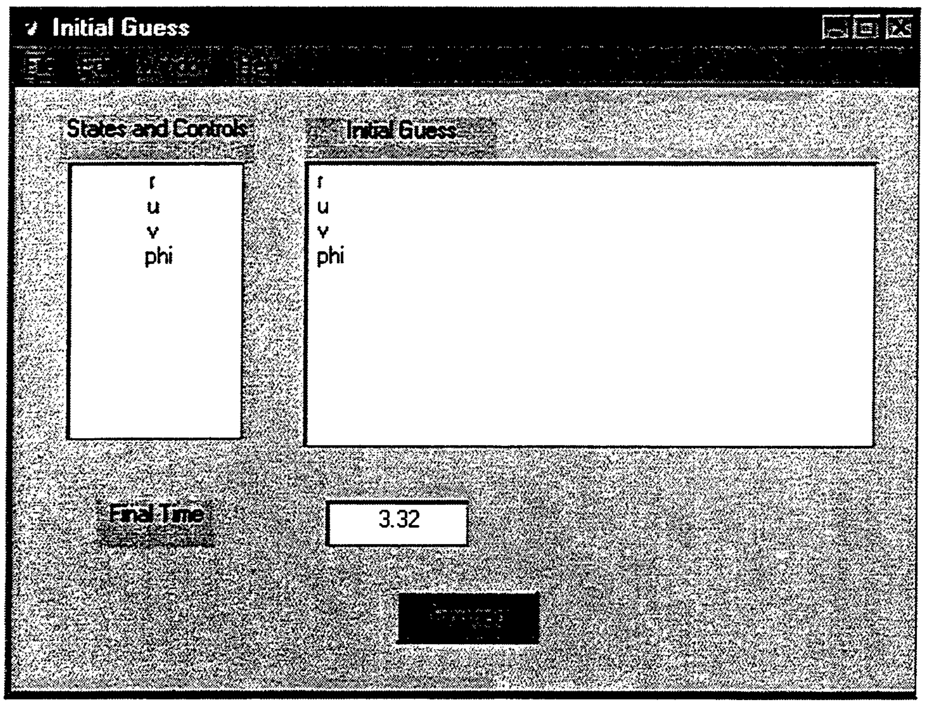

Figure 34. Optimal Time Problem Initial Guess - GUI

button for correctness and the problem is ready for optimization. Pressing the *optimize* button optimizes the problem and appends the results to the last results, see Appendix D. The function file created for a free final time problem no longer has t_f as a parameter passed into the function, but rather assigns the value during run-time. Comparing the two files created for the orbit transfer problem illustrate the subtle differences between the two types, fixed and free final time problems.

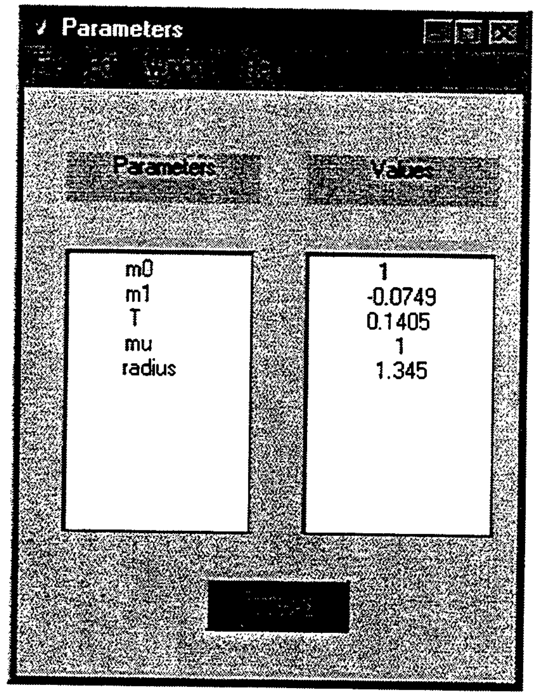

Figure 35. Optimal Time Problem Parameters Values - GUI

VIII. CONCLUSION

The MATLAB GUI for the Legendre-Gauss-Lobatto Pseudospectral algorithm makes solving optimal control problems easier. The GUI has been tailored to solve optimal control problems in MATLAB using the Optimization Toolkit. This provides a wide audience of potential users. Other NLP solvers can be used to solve trajectory optimization problems, such as NPSOL. One of the strengths of MATLAB is its ability to use codes written in non-MATLAB languages such as FORTRAN, C, or C++. The first step to adapting this GUI to a different NLP solver is to understand the inputs that the solver requires. Once the input parameters have been determined, the GUI code can be modified to create different inputs. As most of the code is run from functions, the GUI figures themselves would require little modification. The standard form of an optimal control problem will not change with the addition of additional solvers. The *Optimization* figure would require the most visual modification to add parameters specific to each solver. The GUI could be easily modified to create several input files from the file figure and then allow the user to choose from several solver options.

The GUI can be used to allow students to rapidly solve problems in this area of mathematics. The tool can also be used to provide initial guesses for numerical indirect methods that require very good initial guesses. The GUI has been beta tested by two students from the Aeronautics and Astronautics Department, LT Bryan Schlotman, USN, and Cpt Lawrence Halbach, USAF. Further research using the GUI will be conducted by these two officers here at the Naval Postgraduate School. This GUI will also be used in the Spacecraft Performance and Optimization (AA4850) class in the fall quarter.

The real strength of the LGLP algorithm involves costate estimation from the Lagrange multipliers returned from the NLP solver. The major disappointment with the MATLAB `constr.m` is that is does not return correct Lagrange multipliers.

Adaption of this GUI to use the NPSOL solver will allow the LGLP method to be easily used and evaluated against other methods using the de facto standard sequential quadratic programming solver.

APPENDIX A. CART PROBLEM OPTIMIZATION FILES

This appendix contains one of the files created while solving the simple cart problem of Chapter 6. The following file is the `CONSTR.M` input file which contains the function to be minimized, `costfn`, and the constraint function, g. The file `cart_function.m` follows:

```
function [costfn,g] = cart_function(xopt,Dn,x,w,n,t,Tf,a,b,c);

%Define the variables
x1=xopt( 0*n+1:n* 1)';
x2=xopt( 1*n+1:n* 2)';
u =xopt( 2*n+1:n* 3)';


%Set up the State Constraints
g( 0*n+1:n* 1)=(2/Tf)*(Dn*x1)-(x2   );
g( 1*n+1:n* 2)=(2/Tf)*(Dn*x2)-(-x2+u);


%Set up the initial conditions
g( 2*n+ 1)=x1(1);
g( 2*n+ 2)=x2(2);


%Set up the final time conditions
g( 2*n+ 3)=a*x1(n)+b*x2(n)-c;


%Set up the inequality conditions


%Set up the cost function
costfn=(Tf/2)*(w'*(u.^2));
```

APPENDIX B. MAXIMUM RADIUS PROBLEM OPTIMIZATION FILES

This appendix contains two of the files created while solving the maximum radius orbit transfer problem of Chapter 7. The first file is the `CONSTR.M` input file. The file `bho.m` follows:

```
function [costfn,g] = bho(xopt,Dn,x,w,n,t,Tf,m0,m1,T,mu);

%Define the variables
r  =xopt( 0*n+1:n* 1)';
u  =xopt( 1*n+1:n* 2)';
v  =xopt( 2*n+1:n* 3)';
phi=xopt( 3*n+1:n* 4)';


%Set up the State Constraints
g( 0*n+1:n* 1)=(2/Tf)*(Dn*r  )-(u                                          );
g( 1*n+1:n* 2)=(2/Tf)*(Dn*u  )-((v.^2)./r -mu./(r.^2) + T*sin(phi)./(m0-m1*t));
g( 2*n+1:n* 3)=(2/Tf)*(Dn*v  )-(-(v.*u)./r + T*cos(phi)./(m0-m1*t)          );

%Set up the initial conditions
g( 3*n+ 1)=r(1)-1;
g( 3*n+ 2)=u(1)  ;
g( 3*n+ 3)=v(1)-1;

%Set up the final time conditions
g( 3*n+ 4)=u(n)                ;
g( 3*n+ 5)=v(n)-sqrt(1/r(n))   ;

%Set up the point inequality constraints

%Set up the inequality constraints
g( 3*n+ 6:n* 4+ 5)=phi-pi ;
g( 4*n+ 6:n* 5+ 5)=-phi-pi;

%Set up the cost function
costfn=-r(n);
```

The bho.out file resulting from an optimization run follows:

```
Filename_File =
bho
date =
        1999           6           7          17          56          20
timer =
   50.5900
r =
  Columns 1 through 7
    1.0000    1.0003    1.0033    1.0260    1.0898    1.1794    1.2660
  Columns 8 through 11
    1.3198    1.3397    1.3446    1.3450
u =
  Columns 1 through 7
   -0.0000    0.0046    0.0251    0.1034    0.1709    0.1871    0.1553
  Columns 8 through 11
    0.0821    0.0316    0.0085   -0.0000
v =
  Columns 1 through 7
    1.0000    1.0143    1.0418    1.0651    1.0028    0.8547    0.8158
  Columns 8 through 11
    0.8209    0.8320    0.8533    0.8622
phi =
  Columns 1 through 7
    0.6707   -0.0314    0.5439    0.4999   -3.1416    2.3196   -0.2570
  Columns 8 through 11
   -1.1302   -0.5989   -0.7030   -0.6172
cost =
   -1.3450
```

APPENDIX C. OPTIMAL TIME PROBLEM OPTIMIZATION FILES

This appendix contains two of the files created while solving the optimal time orbit transfer problem of Chapter 7. The first file is the `CONSTR.M` input file. The file `bho.m` follows:

```
function [costfn,g] = bho(xopt,Dn,x,w,n,m0,m1,T,mu,radius);

Tf = xopt(length(xopt));
t = (Tf/2)*(x+1);

%Define the variables
r  =xopt( 0*n+1:n* 1)';
u  =xopt( 1*n+1:n* 2)';
v  =xopt( 2*n+1:n* 3)';
phi=xopt( 3*n+1:n* 4)';

%Set up the State Constraints
g( 0*n+1:n* 1)=(2/Tf)*(Dn*r  )-(u                                              );
g( 1*n+1:n* 2)=(2/Tf)*(Dn*u  )-((v.^2)./r -mu./(r.^2) + T*sin(phi)./(m0-m1*t));
g( 2*n+1:n* 3)=(2/Tf)*(Dn*v  )-(-(v.*u)./r + T*cos(phi)./(m0-m1*t)           );

%Set up the initial conditions
g( 3*n+ 1)=r(1)-1;
g( 3*n+ 2)=u(1)   ;
g( 3*n+ 3)=v(1)-1;

%Set up the final time conditions
g( 3*n+ 4)=u(n)                ;
g( 3*n+ 5)=v(n)-sqrt(1/r(n))   ;
g( 3*n+ 6)=r(n)-radius         ;

%Set up the point inequality constraints
g( 3*n+ 7)=-Tf;

%Set up the inequality constraints
g( 3*n+ 8:n* 4+ 7)=phi-pi ;
g( 4*n+ 8:n* 5+ 7)=-phi-pi;
```

```
%Set up the cost function
costfn=Tf;
```

The bho.out file resulting from an optimization run follows:

```
Filename_File =
bho
date =
        1999            6           18           13           58           27
timer =
   30.6500
r =
  Columns 1 through 7
    1.0000    0.9998    1.0035    1.0245    1.0835    1.1868    1.2781
  Columns 8 through 11
    1.3237    1.3408    1.3447    1.3450
u =
  Columns 1 through 7
    0.0000   -0.0003    0.0335    0.0899    0.1884    0.2294    0.1488
  Columns 8 through 11
    0.0708    0.0288    0.0059    0.0000
v =
  Columns 1 through 7
    1.0000    1.0101    1.0346    1.0540    1.0153    0.8723    0.7967
  Columns 8 through 11
    0.8102    0.8318    0.8520    0.8623
phi =
  Columns 1 through 7
   -1.2427    0.5160    0.6096    0.5087    1.9920    3.1416   -0.9890
  Columns 8 through 11
   -0.5805   -0.6466   -0.5589   -0.3357
cost =
    3.1685
```

APPENDIX D. UTILITY FUNCTIONS

This appendix contains the utility function declare.m. This file is used within the LGLP GUI.

```
function [valstring,count] = declare(vars,vals);
%DECLARE creates variables with the values provided in the
%MATLAB workspace (H1 line)
%
%declare.m        - A utility function for the LGLP GUI
%
%           [valstring,count] = declare(vars,vals)
%
%input  vars      - A vector of strings containing the names of variables
%
%       vals      - A vector of strings containing the values to be assigned
%                   to the input variables
%
%output valstring - A vector of strings that when executed declare variables
%                   in the the MATLAB workspace. The command 'eval(valstring)'
%                   will execute  each string in the vector.
%       count     - A counter for the number of strings or size of valstring
%
%
%written by Andrew O. Hall, CPT, US Army, June 1999, at the
%Naval Postgraduate School

equal = '=';
semi = ';';
count = 1;
valstring = [];
for index = vars'
  vstring = strcat(vars(count,:),equal);
  vstring = strcat(vstring,vals(count,:));
  vstring = strcat(vstring,semi);
  valstring = strvcat(valstring,vstring);
  count = count + 1;
end
count = count - 1;
```

APPENDIX E. CONTENTS FILE

This appendix contains the `Contents.m` file for the LGLP GUI. A file like this needs to be placed within a directory to allow the file to be accessed through the MATLAB help menus.

```
% LGLP Toolkit for Optimal Control Problems
% written by Andrew O. Hall, Naval Postgraduate School
% version 1.0 June 1999
%
% This directory contains a GUI for solving optimal control problems.
% There are three files that execute the GUI.  Typing 'opt' at the command
% line will start the GUI.
%
% GUI Files
%
% OPT.M        - Welcome screen for the GUI interface
%
% FILE.M       - File creation GUI.  Creates an NLP formulation for input to
%                CONSTR.M (see OPTIMIZATION TOOLKIT)
%
% OPTIMIZE.M - File to optimize an optimal control problem.  This
%                function sets all input parameters for CONSTR.M.
%
%
% Utility Files
%
% costcall.m  -   Creates a function call to compute the value the cost
%                 function
%
% declare.m   -   Creates a function call to create variables with the values
%                 provided in the MATLAB workspace
%
% optcall.m   -   Creates a function call for CONSTR.M
%
%
% Example Files - see AOExamples.ps in this directory
%
% carttest.m    -   Simple cart problem, linear constraints with quadratic
%                   controls, from Chapter 6 of Hall's Thesis
```

```
% bh.m           - Maximum Radius Orbit Transfer problem from Chapter 7
%                  of Hall's Thesis
% bho.m          - Optimal Time Orbit Transfer problem from Chapter 7
%                  of Hall's Thesis

% Faculty Codes  - Codes written by Professors Bill Gragg and Fariba Fahroo,
%                  Naval Postgraduate School
%
% diffmat.m      -  creates a differentiation matrix for the LGLP algorithm
% lin_intp.m     -  creates a linear guess vector
% lobatto.m      -  Computes abscissa and weights for the n-point
%                   Gauss-Jacobi-Lobatto quadrature formula
% mxt.m          -  creates a tridiagonal matrix
% mxtj.m         -  creates a Jacobi matrix
% sgn.m          -  For z a complex number we define sgn z, the SIGNUM of z,
%                   as z/|z| if z ~= 0 and + 1 if z = 0.  Thus sgn z is the
%                   same as matlab's sign z except when z = 0.
% tqr.m          -  Tridiagonal QR algorithm

% Postscript Files
%
% AOExamples.ps  -  PS file that includes Chapters 5,6,and 7 from Hall's Thesis
% AOThesis.ps    -  Hall's Naval Postgraduate School Thesis (Applied Mathematics)

% written by Andrew O. Hall, CPT, US Army, June 1999, at the
% Naval Postgraduate School
% Questions?  Email:  AndrewOHall@msn.com
```

LIST OF REFERENCES

[1] Wan, Frederick Y.M., *Introduction to the Calculus of Variations and its Applications*, Chapman & Hall, New York, 1995.

[2] Kirk, Donald E., *An Introduction to Optimal Control*, Prentice-Hall Inc., Englewood Cliffs, 1970.

[3] Betts, John T.,"Survey of Numerical Methods for Trajectory Optimization", *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 2, 1998.

[4] Elnagar, Gamal, Mohammad A. Kazemi, and Mohsen Razzagi,"The Pseudospectral Legendre Method for Discretizing Optimal Control Problems", *IEEE Transactions on Automatic Control* Vol. 40, No. 10, 1995, pp. 1793-1796.

[5] Fahroo, Fariba, and I. Michael Ross."Costate Estimation by a Legendre Pseudospectral Method" *Proceedings of the 1998 AIAA GNC Conference*, Boston, MA, August 1998, AIAA Paper 98-4222.

[6] Stengel, Robert F.,*Optimal Control and Estimation*, Dover Publications, Inc., Mineola, 1986.

[7] Hull, David G.,"Conversion of Optimal Control Problems into Parameter Optimization Problems", *Journal of Guidance, Control, and Dynamics*, Vol. 20, No. 1, 1997.

[8] Conway, B.A. and K. M. Larson,"Collocation Versus Differential Inclusion in Direct Optimization", *Journal of Guidance, Control, and Dynamics*, Vol. 21, No. 5, 1998.

[9] Canuto, C., Hussaini, M.Y., Quarteroni, A., and Zang T. A., *Spectral Methods in Fluid Dynamics*, Springer Verlag, New York, 1988.

[10] Freud, Geza, *Orthogonal Polynomials*, Pergamon Press Ltd.,Oxford, 1971.

[11] Szego, Gabor, *Orthogonal Polynomials*, American Mathematical Society, New York, 1959.

[12] Herman, Albert L., and Bruce A. Conway, "Direct Optimization Using Collocation Based upon High-order Gauss-Lobatto Quadrature Rules", *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 3, 1996.

[13] Marchand, Patrick, *Graphics and GUIs with MATLAB*, CRC Press, New York, 1999.

[14] Bryson, Arthur E. and Yu-Chi Ho, *Applied Optimal Control*, John Wiley & Sons, New York, 1975.

[15] Mathworks, *MATLAB Graphics and GUIs*, 1998.

[16] Mathworks, *Optimization Toolbox*, 1998.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5000

3. Colonel David C. Arney 1
   United States Military Academy
   Department of Mathematical Sciences
   West Point, NY 10996

4. Chairman Michael A. Morgan 1
   Code MA/Mw
   Department of Mathematics
   Naval Postgraduate School
   Monterey, CA 93943-5101

5. Fariba Fahroo 6
   Code MA/Ff
   Department of Mathematics
   Naval Postgraduate School
   Monterey, CA 93943-5101

6. I. Michael Ross 1
   Code AA/Ro
   Department of Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, CA 93943-5101

7. Captain Andrew O. Hall 2
   United States Military Academy
   Department of Mathematical Sciences
   West Point, NY 10996